MQL5 Language REFERENCE

for the MetaTrader 5 client terminal

STUDY MQL5 and SOLVE any tasks:

- Create your own technical indicators of any complexity level
- Use algorithmic trading — automate strategies to work on various financial markets
- Develop your own analytical tools based on mathematical achievements and traditional methods
- Write automated trading systems for solving a wide range of tasks (trading, monitoring, alerting, etc.)

Content

MQL5 Reference 72

1 Language Basics ... 75
- Syntax ... 76
  - Comments ... 77
  - Identifiers ... 78
  - Reserved Words ... 79
- Data Types ... 81
  - Integer Types ... 82
    - Char, Short, Int and Long Types ... 84
    - Character Constants ... 88
    - Datetime Type ... 91
    - Color Type ... 92
    - Bool Type ... 93
    - Enumerations ... 94
  - Real Types (double, float) ... 96
  - Complex number (complex) ... 102
  - String Type ... 103
  - Structures, Classes and Interfaces ... 106
  - Dynamic Array Object ... 133
  - Matrices and vectors ... 134
  - Typecasting ... 141
  - Void Type and NULL Constant ... 146
  - User-defined Types ... 147
  - Object Pointers ... 157
  - References: Modifier & and Keyword this ... 161
- Operations and Expressions ... 163
  - Expressions ... 164
  - Arithmetic Operations ... 165
  - Assignment Operations ... 166
  - Operations of Relation ... 167
  - Boolean Operations ... 168
  - Bitwise Operations ... 170
  - Other Operations ... 173
  - Precedence Rules ... 177
- Operators ... 179
  - Compound Operator ... 181
  - Expression Operator ... 182
  - Return Operator ... 183
  - Conditional Operator if-else ... 184
  - Ternary Operator ?: ... 185
  - Switch Operator ... 187
  - Loop Operator while ... 189
  - Loop Operator for ... 190
  - Loop Operator do while ... 191
  - Break Operator ... 192
  - Continue Operator ... 193
  - Matrix product operator ... 194
  - Object Create Operator new ... 196
  - Object Delete Operator delete ... 197
- Functions ... 198
  - Function Call ... 200
  - Passing Parameters ... 201
  - Function Overloading ... 204

Operation Overloading 207
Description of External Functions 220
Exporting Functions 222
Event Handling Functions 223
Variables **235**
Local Variables 239
Formal Parameters 241
Static Variables 243
Global Variables 245
Input Variables 246
Extern Variables 252
Initialization of Variables 253
Visibility Scope and Lifetime of Variables 255
Creating and Deleting Objects 257
Preprocessor **260**
Macro substitution (#define) 262
Program Properties (#property) 265
Including Files (#include) 271
Importing Functions (#import) 272
Conditional Compilation (#ifdef, #ifndef, #else, #endif) 275
Object-Oriented Programming **277**
Encapsulation and Extensibility of Types 279
Inheritance 282
Polymorphism 287
Overload 291
Virtual Functions 292
Static Members of a Class 296
Function templates 300
Class templates 304
Abstract Classes 309
Namespaces **311**
2 Constants, Enumerations and Structures **314**
Chart Constants **315**
Types of Chart Events 316
Chart Timeframes 323
Chart Properties 325
Positioning Constants 333
Chart Representation 334
Examples of Working with the Chart 336
Objects Constants **394**
Object Types 395
OBJ_VLINE 397
OBJ_HLINE 402
OBJ_TREND 407
OBJ_TRENDBYANGLE 414
OBJ_CYCLES 420
OBJ_ARROWED_LINE 426
OBJ_CHANNEL 432
OBJ_STDDEVCHANNEL 439
OBJ_REGRESSION 446
OBJ_PITCHFORK 452
OBJ_GANNLINE 460
OBJ_GANNFAN 467
OBJ_GANNGRID 474
OBJ_FIBO 481
OBJ_FIBOTIMES 488
OBJ_FIBOFAN 495
OBJ_FIBOARC 502

OBJ_FIBOCHANNEL 509
OBJ_EXPANSION 517
OBJ_ELLIOTWAVE5 525
OBJ_ELLIOTWAVE3 533
OBJ_RECTANGLE 540
OBJ_TRIANGLE 546
OBJ_ELLIPSE 553
OBJ_ARROW_THUMB_UP 559
OBJ_ARROW_THUMB_DOWN 565
OBJ_ARROW_UP 571
OBJ_ARROW_DOWN 577
OBJ_ARROW_STOP 583
OBJ_ARROW_CHECK 589
OBJ_ARROW_LEFT_PRICE 595
OBJ_ARROW_RIGHT_PRICE 600
OBJ_ARROW_BUY 605
OBJ_ARROW_SELL 610
OBJ_ARROW 615
OBJ_TEXT 621
OBJ_LABEL 627
OBJ_BUTTON 635
OBJ_CHART 642
OBJ_BITMAP 649
OBJ_BITMAP_LABEL 656
OBJ_EDIT 663
OBJ_EVENT 670
OBJ_RECTANGLE_LABEL 675
Object Properties 681
Methods of Object Binding 708
Chart Corner 713
Visibility of Objects 716
Levels of Elliott Wave 719
Gann Objects 720
Web Colors 722
Wingdings 724
Indicator Constants 725
Price Constants 726
Smoothing Methods 729
Indicators Lines 730
Drawing Styles 732
Custom Indicator Properties 737
Indicator Types 743
Data Type Identifiers 745
Environment State 746
Client Terminal Properties 747
Running MQL5 Program Properties 757
Symbol Properties 761
Account Properties 874
Testing Statistics 884
Trade Constants 889
History Database Properties 890
Order Properties 891
Position Properties 909
Deal Properties 913
Trade Operation Types 917
Trade Transaction Types 929
Trade Orders in DOM 931
Signal Properties 932

Named Constants 934
Predefined Macro Substitutions 935
Mathematical Constants 941
Numerical Type Constants 943
Uninitialization Reason Codes 946
Checking Object Pointer 948
Other Constants 949
Data Structures 953
Date Type Structure 954
Indicator Parameter Structure 955
History Data Structure 956
Order Book Structure 957
Trade Request Structure 958
Request Check Result Structure 971
Trade Request Result Structure 972
Trade Transaction Structure 975
Price Data Structure 983
Economic Calendar structures 985
Codes of Errors and Warnings 994
Trade Server Return Codes 995
Compiler Warnings 998
Compilation Errors 1001
Runtime Errors 1012
Input/Output Constants 1025
File Opening Flags 1026
File Properties 1028
In-File Position 1029
Use of a Codepage 1030
MessageBox 1031
3 MQL5 programs 1033
Program Running 1034
Trade Permission 1041
Client Terminal Events 1045
Resources 1048
Call of Imported Functions 1060
Runtime Errors 1062
Testing Trading Strategies 1063
4 Predefined Variables 1089
_AppliedTo 1090
_Digits 1092
_Point 1093
_LastError 1094
_Period 1095
_RandomSeed 1096
_StopFlag 1097
_Symbol 1098
_UninitReason 1099
_IsX64 1100
5 Common Functions 1101
Alert 1103
CheckPointer 1108
Comment 1110
CryptEncode 1111
CryptDecode 1113
DebugBreak 1115
ExpertRemove 1116
GetPointer 1118

GetTickCount 1122
GetTickCount64 1123
GetMicrosecondCount 1125
MessageBox 1127
PeriodSeconds 1132
PlaySound 1134
Print 1137
PrintFormat 1139
ResetLastError 1145
ResourceCreate 1146
ResourceFree 1150
ResourceReadImage 1153
ResourceSave 1157
SetReturnError 1162
SetUserError 1164
Sleep 1165
TerminalClose 1167
TesterHideIndicators 1169
TesterStatistics 1171
TesterStop 1173
TesterDeposit 1175
TesterWithdrawal 1178
TranslateKey 1181
ZeroMemory 1182
6 Array Functions 1185
ArrayBsearch 1187
ArrayCopy 1191
ArrayCompare 1196
ArrayFree 1199
ArrayGetAsSeries 1208
ArrayInitialize 1211
ArrayFill 1213
ArrayIsDynamic 1215
ArrayIsSeries 1217
ArrayMaximum 1219
ArrayMinimum 1230
ArrayPrint 1241
ArrayRange 1244
ArrayResize 1245
ArrayInsert 1248
ArrayRemove 1251
ArrayReverse 1253
ArraySetAsSeries 1255
ArraySize 1258
ArraySort 1260
ArraySwap 1265
ArrayToFP16 1267
ArrayToFP8 1270
ArrayFromFP16 1273
ArrayFromFP8 1276
7 Matrix and Vector Methods 1279
Matrix and Vector Types 1286
    Enumerations 1287
Initialization 1292
    Assign 1295
    CopyIndicatorBuffer 1297
    CopyRates 1299
    CopyTicks 1304

CopyTicksRange 1307
Eye 1309
Identity 1311
Ones 1313
Zeros 1314
Full 1315
Tri 1316
Init 1317
Fill 1319
Random 1320

Manipulations 1322

HasNan 1324
Transpose 1325
TransposeConjugate 1327
TriL 1328
TriU 1329
Diag 1330
Row 1332
Col 1334
Copy 1336
Concat 1338
Compare 1340
CompareByDigits 1342
CompareEqual 1344
Flat 1347
Clip 1349
Reshape 1350
Resize 1352
Set 1354
SwapRows 1356
SwapCols 1357
Split 1358
Hsplit 1360
Vsplit 1362
ArgSort 1364
Sort 1365

Operations 1367

Mathematical operations 1369
Mathematical functions 1370

Products 1371

MatMul 1372
GeMM 1377
Power 1381
Dot 1384
Kron 1386
Inner 1388
Outer 1390
CorrCoef 1393
Cov 1397
Correlate 1400
Convolve 1403

Transformations 1406

Cholesky 1407
Eig 1408
EigVals 1411
LU 1412
LUP 1414
QR 1416

SVD 1418

Statistics **1421**

ArgMax 1422

ArgMin 1423

Max 1424

Min 1425

Ptp 1426

Sum 1427

Prod 1428

CumSum 1430

CumProd 1432

Percentile 1434

Quantile 1436

Median 1438

Mean 1440

Average 1441

Std 1443

Var 1445

LinearRegression 1447

Features **1450**

Rows 1451

Cols 1452

Size 1453

Norm 1454

Cond 1457

Det 1460

SLogDet 1462

Rank 1463

Trace 1466

Spectrum 1467

Solutions **1468**

Solve 1469

LstSq 1470

Inv 1471

PInv 1473

Machine learning **1474**

Activation 1479

Derivative 1483

Loss 1485

LossGradient 1487

RegressionMetric 1489

ConfusionMatrix 1491

ConfusionMatrixMultilabel 1493

ClassificationMetric 1495

ClassificationScore 1499

PrecisionRecall 1503

ReceiverOperatingCharacteristic 1507

OpenBLAS **1511**

Singular Value Decomposition 1514

SingularValueDecompositionDC 1515

SingularValueDecompositionQR 1517

SingularValueDecompositionQRPivot 1519

SingularValueDecompositionBisect 1523

SingularValueDecompositionJacobiHigh 1526

SingularValueDecompositionJacobiLow 1531

SingularValueDecompositionBidiagDC 1534

SingularValueDecompositionBidiagBisect 1537

SingularValueDecompositionBidiagQR 1540

- Eigen Values ... 1542
  - General Matrices ... 1545
    - EigenSolver ... 1546
    - EigenSolverX ... 1548
    - EigenSolverShur ... 1552
    - EigenSolver2 ... 1554
    - EigenSolver2X ... 1557
    - EigenSolver2Shur ... 1562
    - EigenSolver2Blocked ... 1565
    - EigenSolver2ShurBlocked ... 1568
  - Symmetric Matrices ... 1571
    - EigenSymmetricDC ... 1573
    - EigenSymmetricQR ... 1575
    - EigenSymmetricRobust ... 1577
    - EigenSymmetricBisect ... 1580
    - EigenSymmetricDC2s ... 1583
    - EigenSymmetricQR2s ... 1585
    - EigenSymmetricRobust2s ... 1587
    - EigenSymmetricBisect2s ... 1590
    - EigenSymmetric2DC ... 1593
    - EigenSymmetric2QR ... 1596
    - EigenSymmetric2Bisect ... 1599
  - Tridiagonal Matrices ... 1603
    - EigenTridiagonalDC ... 1605
    - EigenTridiagonalQR ... 1607
    - EigenTridiagonalRobust ... 1609
    - EigenTridiagonalBisect ... 1612
    - EigenTridiagonalQL ... 1615
    - EigenTridiagonalDCQ ... 1616
    - EigenTridiagonalQRQ ... 1618
    - EigenTridiagonalPosDefQ ... 1620
- Orthogonal Factorizations ... 1622
  - FactorizationQR ... 1623
  - FactorizationQRNonNeg ... 1625
  - FactorizationQRPivot ... 1627
  - FactorizationLQ ... 1629
  - FactorizationQL ... 1631
  - FactorizationRQ ... 1633
- Linear Equations ... 1635
  - LinearEquationsSolution ... 1637
  - LinearEquationsSolutionTriangular ... 1639
  - LinearEquationsSolutionSy ... 1641
  - LinearEquationsSolutionComplexSy ... 1643
  - LinearEquationsSolutionSyPD ... 1644
  - LinearEquationsSolutionGeTrid ... 1646
  - LinearEquationsSolutionSyTridPD ... 1648
- Factorizations ... 1650
  - FactorizationPLU ... 1651
  - FactorizationPLUGeTrid ... 1653
  - FactorizationLDL ... 1655
  - FactorizationLDLSyTridPD ... 1657
  - FactorizationCholesky ... 1659
  - FactorizationCholeskySyPS ... 1660
- Dynamic Mode Decomposition ... 1662
  - DynamicModeDecomposition ... 1663
  - DynamicModeDecompositionQR ... 1669
- Singular Spectrum Analysis ... 1676
  - SingularSpectrumAnalysisSpectrum ... 1677

SingularSpectrumAnalysisForecast ... 1678
SingularSpectrumAnalysisReconstructComponents ... 1679
SingularSpectrumAnalysisReconstructSeries ... 1680
Reductions ... 1681
ReduceToBidiagonal ... 1682
ReflectBidiagonalToQP ... 1685
ReduceSymmetricToTridiagonal ... 1687
ReflectTridiagonalToQ ... 1689
8 Conversion Functions ... 1691
CharToString ... 1693
CharArrayToString ... 1695
CharArrayToStruct ... 1697
StructToCharArray ... 1701
ColorToARGB ... 1704
ColorToString ... 1706
DoubleToString ... 1707
EnumToString ... 1708
IntegerToString ... 1710
ShortToString ... 1712
ShortArrayToString ... 1714
TimeToString ... 1716
NormalizeDouble ... 1719
StringToCharArray ... 1721
StringToColor ... 1723
StringToDouble ... 1724
StringToInteger ... 1725
StringToShortArray ... 1726
StringToTime ... 1728
StringFormat ... 1731
9 Math Functions ... 1735
MathAbs ... 1737
MathArccos ... 1738
MathArcsin ... 1741
MathArctan ... 1744
MathArctan2 ... 1747
MathClassify ... 1751
MathCeil ... 1753
MathCos ... 1755
MathExp ... 1758
MathFloor ... 1761
MathLog ... 1763
MathLog10 ... 1767
MathMax ... 1771
MathMin ... 1773
MathMod ... 1775
MathPow ... 1776
MathRand ... 1780
MathRound ... 1781
MathSin ... 1783
MathSqrt ... 1786
MathSrand ... 1790
MathTan ... 1793
MathIsValidNumber ... 1796
MathExpm1 ... 1797
MathLog1p ... 1800
MathArccosh ... 1804
MathArcsinh ... 1807
MathArctanh ... 1810

MathCosh 1813
MathSinh 1816
MathTanh 1819
MathSwap 1822
10 String Functions 1825
StringAdd 1826
StringBufferLen 1828
StringCompare 1829
StringConcatenate 1831
StringFill 1833
StringFind 1834
StringGetCharacter 1837
StringInit 1839
StringLen 1840
StringSetLength 1841
StringReplace 1842
StringReserve 1843
StringSetCharacter 1845
StringSplit 1847
StringSubstr 1849
StringToLower 1850
StringToUpper 1851
StringTrimLeft 1852
StringTrimRight 1853
11 Date and Time 1854
TimeCurrent 1855
TimeTradeServer 1857
TimeLocal 1859
TimeGMT 1861
TimeDaylightSavings 1863
TimeGMTOffset 1864
TimeToStruct 1865
StructToTime 1866
12 Account Information 1868
AccountInfoDouble 1869
AccountInfoInteger 1870
AccountInfoString 1872
13 Checkup 1873
GetLastError 1874
IsStopped 1875
UninitializeReason 1876
TerminalInfoInteger 1878
TerminalInfoDouble 1879
TerminalInfoString 1880
MQLInfoInteger 1881
MQLInfoString 1882
Symbol 1883
Period 1884
Digits 1885
Point 1886
14 Event Handling 1887
OnStart 1889
OnInit 1892
OnDeinit 1895
OnTick 1898
OnCalculate 1904

OnTimer 1908
OnTrade 1911
OnTradeTransaction 1916
OnBookEvent 1922
OnChartEvent 1925
OnTester 1932
OnTesterInit 1939
OnTesterDeinit 1946
OnTesterPass 1947
15 Market Info 1948
SymbolsTotal 1949
SymbolExist 1950
SymbolName 1952
SymbolSelect 1954
SymbolIsSynchronized 1956
SymbolInfoDouble 1957
SymbolInfoInteger 1959
SymbolInfoString 1961
SymbolInfoMarginRate 1963
SymbolInfoTick 1965
SymbolInfoSessionQuote 1966
SymbolInfoSessionTrade 1968
MarketBookAdd 1970
MarketBookRelease 1972
MarketBookGet 1974
16 Economic Calendar 1975
CalendarCountryById 1976
CalendarEventById 1978
CalendarValueById 1981
CalendarCountries 1984
CalendarEventByCountry 1986
CalendarEventByCurrency 1988
CalendarValueHistoryByEvent 1990
CalendarValueHistory 1993
CalendarValueLastByEvent 1996
CalendarValueLast 2001
17 Timeseries and Indicators Access 2006
Indexing Direction in Arrays, Buffers and Timeseries 2011
Organizing Data Access 2014
SeriesInfoInteger 2023
Bars 2025
BarsCalculated 2028
IndicatorCreate 2030
IndicatorParameters 2032
IndicatorRelease 2034
CopyBuffer 2036
CopyRates 2041
CopySeries 2045
CopyTime 2049
CopyOpen 2052
CopyHigh 2055
CopyLow 2059
CopyClose 2062
CopyTickVolume 2065
CopyRealVolume 2069
CopySpread 2072
CopyTicks 2076

CopyTicksRange 2082
iBars 2084
iBarShift 2085
iClose 2088
iHigh 2090
iHighest 2092
iLow 2093
iLowest 2095
iOpen 2096
iTime 2098
iTickVolume 2100
iRealVolume 2102
iVolume 2104
iSpread 2106
18 Custom Symbols.... 2108
CustomSymbolCreate 2109
CustomSymbolDelete 2112
CustomSymbolSetInteger 2114
CustomSymbolSetDouble 2118
CustomSymbolSetString 2122
CustomSymbolSetMarginRate 2126
CustomSymbolSetSessionQuote 2131
CustomSymbolSetSessionTrade 2136
CustomRatesDelete 2141
CustomRatesReplace 2146
CustomRatesUpdate 2151
CustomTicksAdd 2156
CustomTicksDelete 2164
CustomTicksReplace 2172
CustomBookAdd 2181
19 Chart Operations.... 2184
ChartApplyTemplate 2186
ChartSaveTemplate 2189
ChartWindowFind 2194
ChartTimePriceToXY 2196
ChartXYToTimePrice 2198
ChartOpen 2200
ChartFirst 2202
ChartNext 2203
ChartClose 2204
ChartSymbol 2206
ChartPeriod 2207
ChartRedraw 2208
ChartSetDouble 2211
ChartSetInteger 2213
ChartSetString 2215
ChartGetDouble 2217
ChartGetInteger 2219
ChartGetString 2221
ChartNavigate 2223
ChartID 2226
ChartIndicatorAdd 2228
ChartIndicatorDelete 2232
ChartIndicatorGet 2235
ChartIndicatorName 2237
ChartIndicatorsTotal 2239
ChartWindowOnDropped 2241
ChartPriceOnDropped 2242

ChartTimeOnDropped 2243
ChartXOnDropped 2244
ChartYOnDropped 2245
ChartSetSymbolPeriod 2246
ChartScreenShot 2248
20 Trade Functions 2251
OrderCalcMargin 2253
OrderCalcProfit 2257
OrderCheck 2260
OrderSend 2264
OrderSendAsync 2269
PositionsTotal 2280
PositionGetSymbol 2281
PositionSelect 2283
PositionSelectByTicket 2285
PositionGetDouble 2287
PositionGetInteger 2290
PositionGetString 2292
PositionGetTicket 2295
OrdersTotal 2297
OrderGetTicket 2298
OrderSelect 2300
OrderGetDouble 2304
OrderGetInteger 2308
OrderGetString 2314
HistorySelect 2317
HistorySelectByPosition 2319
HistoryOrderSelect 2323
HistoryOrdersTotal 2326
HistoryOrderGetTicket 2327
HistoryOrderGetDouble 2329
HistoryOrderGetInteger 2333
HistoryOrderGetString 2336
HistoryDealSelect 2339
HistoryDealsTotal 2342
HistoryDealGetTicket 2343
HistoryDealGetDouble 2345
HistoryDealGetInteger 2350
HistoryDealGetString 2353
21 Trade Signals 2357
SignalBaseGetDouble 2358
SignalBaseGetInteger 2359
SignalBaseGetString 2360
SignalBaseSelect 2361
SignalBaseTotal 2362
SignalInfoGetDouble 2363
SignalInfoGetInteger 2364
SignalInfoGetString 2365
SignalInfoSetDouble 2366
SignalInfoSetInteger 2367
SignalSubscribe 2368
SignalUnSubscribe 2369
22 Network Functions 2370
SocketCreate 2372
SocketClose 2375
SocketConnect 2378
SocketIsConnected 2382

SocketIsReadable 2386
SocketIsWritable 2389
SocketTimeouts 2393
SocketRead 2395
SocketSend 2399
SocketTlsHandshake 2403
SocketTlsCertificate 2408
SocketTlsRead 2412
SocketTlsReadAvailable 2416
SocketTlsSend 2421
WebRequest 2425
SendFTP 2428
SendMail 2430
SendNotification 2432
23 Global Variables of the Terminal 2434
GlobalVariableCheck 2435
GlobalVariableTime 2436
GlobalVariableDel 2437
GlobalVariableGet 2438
GlobalVariableName 2439
GlobalVariableSet 2440
GlobalVariablesFlush 2441
GlobalVariableTemp 2442
GlobalVariableSetOnCondition 2443
GlobalVariablesDeleteAll 2444
GlobalVariablesTotal 2445
24 File Functions 2446
FileSelectDialog 2449
FileFindFirst 2451
FileFindNext 2453
FileFindClose 2455
FileIsExist 2457
FileOpen 2460
FileClose 2463
FileCopy 2464
FileDelete 2467
FileMove 2469
FileFlush 2471
FileGetInteger 2473
FileIsEnding 2476
FileIsLineEnding 2478
FileReadArray 2483
FileReadBool 2485
FileReadDatetime 2488
FileReadDouble 2491
FileReadFloat 2494
FileReadInteger 2497
FileReadLong 2501
FileReadNumber 2504
FileReadString 2509
FileReadStruct 2511
FileSeek 2515
FileSize 2518
FileTell 2520
FileWrite 2523
FileWriteArray 2526
FileWriteDouble 2529
FileWriteFloat 2532

FileWriteInteger 2534
FileWriteLong 2537
FileWriteString 2539
FileWriteStruct 2542
FileLoad 2545
FileSave 2547
FolderCreate 2549
FolderDelete 2552
FolderClean 2555
25 Custom Indicators 2558
Indicator Styles in Examples 2561
DRAW_NONE 2572
DRAW_LINE 2575
DRAW_SECTION 2579
DRAW_HISTOGRAM 2583
DRAW_HISTOGRAM2 2587
DRAW_ARROW 2591
DRAW_ZIGZAG 2596
DRAW_FILLING 2601
DRAW_BARS 2606
DRAW_CANDLES 2612
DRAW_COLOR_LINE 2618
DRAW_COLOR_SECTION 2623
DRAW_COLOR_HISTOGRAM 2629
DRAW_COLOR_HISTOGRAM2 2634
DRAW_COLOR_ARROW 2639
DRAW_COLOR_ZIGZAG 2645
DRAW_COLOR_BARS 2650
DRAW_COLOR_CANDLES 2657
Connection between Indicator Properties and Functions 2664
SetIndexBuffer 2667
IndicatorSetDouble 2670
IndicatorSetInteger 2674
IndicatorSetString 2678
PlotIndexSetDouble 2681
PlotIndexSetInteger 2682
PlotIndexSetString 2686
PlotIndexGetInteger 2687
26 Object Functions 2690
ObjectCreate 2692
ObjectName 2696
ObjectDelete 2697
ObjectsDeleteAll 2698
ObjectFind 2699
ObjectGetTimeByValue 2700
ObjectGetValueByTime 2701
ObjectMove 2702
ObjectsTotal 2703
ObjectSetDouble 2704
ObjectSetInteger 2708
ObjectSetString 2711
ObjectGetDouble 2713
ObjectGetInteger 2715
ObjectGetString 2717
TextSetFont 2719
TextOut 2721
TextGetSize 2725

27 Technical Indicators 2726
iAC 2729
iAD 2734
iADX 2739
iADXWilder 2744
iAlligator 2749
iAMA 2756
iAO 2761
iATR 2766
iBearsPower 2771
iBands 2776
iBullsPower 2782
iCCI 2787
iChaikin 2792
iCustom 2797
iDEMA 2801
iDeMarker 2806
iEnvelopes 2811
iForce 2817
iFractals 2822
iFrAMA 2827
iGator 2832
iIchimoku 2839
iBWMFI 2846
iMomentum 2851
iMFI 2856
iMA 2861
iOsMA 2866
iMACD 2871
iOBV 2877
iSAR 2882
iRSI 2887
iRVI 2892
iStdDev 2897
iStochastic 2902
iTEMA 2908
iTriX 2913
iWPR 2918
iVIDyA 2923
iVolumes 2928
28 Working with Optimization Results 2933
FrameFirst 2935
FrameFilter 2936
FrameNext 2937
FrameInputs 2938
FrameAdd 2939
ParameterGetRange 2940
ParameterSetRange 2943
29 Working with Events 2945
EventSetMillisecondTimer 2946
EventSetTimer 2947
EventKillTimer 2948
EventChartCustom 2949
30 Working with OpenCL 2955
CLHandleType 2957
CLGetInfoInteger 2958
CLGetInfoString 2961

CLContextCreate 2964
CLContextFree 2965
CLGetDeviceInfo 2966
CLProgramCreate 2971
CLProgramFree 2976
CLKernelCreate 2977
CLKernelFree 2978
CLSetKernelArg 2979
CLSetKernelArgMem 2980
CLSetKernelArgMemLocal 2981
CLBufferCreate 2982
CLBufferFree 2983
CLBufferWrite 2984
CLBufferRead 2989
CLExecute 2993
CLExecutionStatus 2995
31 Working with databases 2996
DatabaseOpen 2999
DatabaseClose 3001
DatabaseImport 3002
DatabaseExport 3005
DatabasePrint 3011
DatabaseTableExists 3016
DatabaseExecute 3017
DatabasePrepare 3029
DatabaseReset 3038
DatabaseBind 3044
DatabaseBindArray 3049
DatabaseRead 3054
DatabaseReadBind 3055
DatabaseFinalize 3059
DatabaseTransactionBegin 3060
DatabaseTransactionCommit 3065
DatabaseTransactionRollback 3066
DatabaseColumnsCount 3067
DatabaseColumnName 3068
DatabaseColumnType 3069
DatabaseColumnSize 3070
DatabaseColumnText 3071
DatabaseColumnInteger 3072
DatabaseColumnLong 3073
DatabaseColumnDouble 3074
DatabaseColumnBlob 3075
32 Working with DirectX 3076
DXContextCreate 3078
DXContextSetSize 3079
DXContextGetSize 3080
DXContextClearColors 3081
DXContextClearDepth 3082
DXContextGetColors 3083
DXContextGetDepth 3084
DXBufferCreate 3085
DXTextureCreate 3086
DXInputCreate 3092
DXInputSet 3093
DXShaderCreate 3094
DXShaderSetLayout 3095
DXShaderInputsSet 3096

DXShaderTexturesSet ...3097
DXDraw ...3098
DXDrawIndexed ...3099
DXPrimiveTopologySet ...3100
DXBufferSet ...3101
DXShaderSet ...3102
DXHandleType ...3103
DXRelease ...3104
33 Python Integration ... 3105
initialize ...3111
login ...3113
shutdown ...3116
version ...3117
last_error ...3119
account_info ...3121
terminal_info ...3124
symbols_total ...3127
symbols_get ...3128
symbol_info ...3131
symbol_info_tick ...3135
symbol_select ...3137
market_book_add ...3141
market_book_get ...3142
market_book_release ...3145
copy_rates_from ...3146
copy_rates_from_pos ...3150
copy_rates_range ...3153
copy_ticks_from ...3156
copy_ticks_range ...3159
orders_total ...3162
orders_get ...3163
order_calc_margin ...3166
order_calc_profit ...3169
order_check ...3172
order_send ...3176
positions_total ...3181
positions_get ...3182
history_orders_total ...3185
history_orders_get ...3187
history_deals_total ...3190
history_deals_get ...3192
34 ONNX models ... 3196
ONNX Support ...3197
Format Conversion ...3199
Automatic data type conversion ...3200
Creating a Model ...3204
Running a model ...3211
Validation in the Strategy Tester ...3217
OnnxCreate ...3222
OnnxCreateFromBuffer ...3223
OnnxRelease ...3224
OnnxRun ...3225
OnnxGetInputCount ...3228
OnnxGetOutputCount ...3229
OnnxGetInputName ...3230
OnnxGetOutputName ...3231
OnnxGetInputTypeInfo ...3232
OnnxGetOutputTypeInfo ...3233

OnnxSetInputShape 3234
OnnxSetOutputShape 3235
Data structures 3236

35 Standard Library 3240

Mathematics 3241
Statistics 3242
Statistical Characteristics 3245
MathMean 3246
MathVariance 3247
MathSkewness 3248
MathKurtosis 3249
MathMoments 3250
MathMedian 3251
MathStandardDeviation 3252
MathAverageDeviation 3253
Normal Distribution 3254
MathProbabilityDensityNormal 3258
MathCumulativeDistributionNormal 3260
MathQuantileNormal 3262
MathRandomNormal 3264
MathMomentsNormal 3265
Log-normal distribution 3266
MathProbabilityDensityLognormal 3270
MathCumulativeDistributionLognormal 3272
MathQuantileLognormal 3274
MathRandomLognormal 3276
MathMomentsLognormal 3277
Beta distribution 3278
MathProbabilityDensityBeta 3282
MathCumulativeDistributionBeta 3284
MathQuantileBeta 3286
MathRandomBeta 3288
MathMomentsBeta 3289
Noncentral beta distribution 3290
MathProbabilityDensityNoncentralBeta 3294
MathCumulativeDistributionNoncentralBeta 3296
MathQuantileNoncentralBeta 3298
MathRandomNoncentralBeta 3300
MathMomentsNoncentralBeta 3301
Gamma distribution 3302
MathProbabilityDensityGamma 3306
MathCumulativeDistributionGamma 3308
MathQuantileGamma 3310
MathRandomGamma 3312
MathMomentsGamma 3313
Chi-squared distribution 3314
MathProbabilityDensityChiSquare 3318
MathCumulativeDistributionChiSquare 3320
MathQuantileChiSquare 3322
MathRandomChiSquare 3324
MathMomentsChiSquare 3325
Noncentral chi-squared distribution 3326
MathProbabilityDensityNoncentralChiSquare 3330
MathCumulativeDistributionNoncentralChiSquare 3332
MathQuantileNoncentralChiSquare 3334
MathRandomNoncentralChiSquare 3336
MathMomentsNoncentralChiSquare 3337
Exponential distribution 3338

MathProbabilityDensityExponential 3342
MathCumulativeDistributionExponential 3344
MathQuantileExponential 3346
MathRandomExponential 3348
MathMomentsExponential 3349
F-distribution 3350
MathProbabilityDensityF 3354
MathCumulativeDistributionF 3356
MathQuantileF 3358
MathRandomF 3360
MathMomentsF 3361
Noncentral F-distribution 3362
MathProbabilityDensityNoncentralF 3366
MathCumulativeDistributionNoncentralF 3368
MathQuantileNoncentralF 3370
MathRandomNoncentralF 3372
MathMomentsNoncentralF 3373
T-distribution 3374
MathProbabilityDensityT 3378
MathCumulativeDistributionT 3380
MathQuantileT 3382
MathRandomT 3384
MathMomentsT 3385
Noncentral t-distribution 3386
MathProbabilityDensityNoncentralT 3390
MathCumulativeDistributionNoncentralT 3392
MathQuantileNoncentralT 3394
MathRandomNoncentralT 3396
MathMomentsNoncentralT 3397
Logistic distribution 3398
MathProbabilityDensityLogistic 3402
MathCumulativeDistributionLogistic 3404
MathQuantileLogistic 3406
MathRandomLogistic 3408
MathMomentsLogistic 3409
Cauchy distribution 3410
MathProbabilityDensityCauchy 3414
MathCumulativeDistributionCauchy 3416
MathQuantileCauchy 3418
MathRandomCauchy 3420
MathMomentsCauchy 3421
Uniform distribution 3422
MathProbabilityDensityUniform 3426
MathCumulativeDistributionUniform 3428
MathQuantileUniform 3430
MathRandomUniform 3432
MathMomentsUniform 3433
Weibull distribution 3434
MathProbabilityDensityWeibull 3438
MathCumulativeDistributionWeibull 3440
MathQuantileWeibull 3442
MathRandomWeibull 3444
MathMomentsWeibull 3445
Binomial distribution 3446
MathProbabilityDensityBinomial 3449
MathCumulativeDistributionBinomial 3451
MathQuantileBinomial 3453
MathRandomBinomial 3455

MathMomentsBinomial ... 3456
Negative binomial distribution ... 3457
MathProbabilityDensityNegativeBinomial ... 3460
MathCumulativeDistributionNegativeBinomial ... 3462
MathQuantileNegativeBinomial ... 3464
MathRandomNegativeBinomial ... 3466
MathMomentsNegativeBinomial ... 3467
Geometric distribution ... 3468
MathProbabilityDensityGeometric ... 3472
MathCumulativeDistributionGeometric ... 3474
MathQuantileGeometric ... 3476
MathRandomGeometric ... 3478
MathMomentsGeometric ... 3479
Hypergeometric distribution ... 3480
MathProbabilityDensityHypergeometric ... 3484
MathCumulativeDistributionHypergeometric ... 3486
MathQuantileHypergeometric ... 3488
MathRandomHypergeometric ... 3490
MathMomentsHypergeometric ... 3491
Poisson distribution ... 3492
MathProbabilityDensityPoisson ... 3496
MathCumulativeDistributionPoisson ... 3498
MathQuantilePoisson ... 3500
MathRandomPoisson ... 3502
MathMomentsPoisson ... 3503
Subfunctions ... 3504
MathRandomNonZero ... 3509
MathMoments ... 3510
MathPowInt ... 3511
MathFactorial ... 3512
MathTrunc ... 3513
MathRound ... 3514
MathArctan2 ... 3516
MathGamma ... 3518
MathGammaLog ... 3519
MathBeta ... 3520
MathBetaLog ... 3521
MathBetaIncomplete ... 3522
MathGammaIncomplete ... 3523
MathBinomialCoefficient ... 3524
MathBinomialCoefficientLog ... 3525
MathHypergeometric2F2 ... 3526
MathSequence ... 3527
MathSequenceByCount ... 3528
MathReplicate ... 3529
MathReverse ... 3530
MathIdentical ... 3531
MathUnique ... 3532
MathQuickSortAscending ... 3533
MathQuickSortDescending ... 3534
MathQuickSort ... 3535
MathOrder ... 3536
MathBitwiseNot ... 3537
MathBitwiseAnd ... 3538
MathBitwiseOr ... 3539
MathBitwiseXor ... 3540
MathBitwiseShiftL ... 3541
MathBitwiseShiftR ... 3542

MathCumulativeSum 3543
MathCumulativeProduct 3544
MathCumulativeMin 3545
MathCumulativeMax 3546
MathSin 3547
MathCos 3548
MathTan 3549
MathArcsin 3550
MathArccos 3551
MathArctan 3552
MathSinPi 3553
MathCosPi 3554
MathTanPi 3555
MathAbs 3556
MathCeil 3557
MathFloor 3558
MathSqrt 3559
MathExp 3560
MathPow 3561
MathLog 3562
MathLog2 3563
MathLog10 3564
MathLog1p 3565
MathDifference 3566
MathSample 3568
MathTukeySummary 3571
MathRange 3572
MathMin 3573
MathMax 3574
MathSum 3575
MathProduct 3576
MathStandardDeviation 3577
MathAverageDeviation 3578
MathMedian 3579
MathMean 3580
MathVariance 3581
MathSkewness 3582
MathKurtosis 3583
MathExpm1 3584
MathSinh 3585
MathCosh 3586
MathTanh 3587
MathArcsinh 3588
MathArccosh 3589
MathArctanh 3590
MathSignif 3591
MathRank 3593
MathCorrelationPearson 3594
MathCorrelationSpearman 3595
MathCorrelationKendall 3596
MathQuantile 3597
MathProbabilityDensityEmpirical 3598
MathCumulativeDistributionEmpirical 3599
Fuzzy Logic 3600
Membership functions 3601
CConstantMembershipFunction 3603
GetValue 3605
CCompositeMembershipFunction 3606

CompositionType ... 3608
MembershipFunctions ... 3608
GetValue ... 3608
CDifferencTwoSigmoidalMembershipFunction ... 3610
A1 ... 3612
A2 ... 3612
C1 ... 3613
C2 ... 3613
GetValue ... 3614
CGeneralizedBellShapedMembershipFunction ... 3615
A ... 3617
B ... 3617
C ... 3618
GetValue ... 3618
CNormalCombinationMembershipFunction ... 3619
B1 ... 3621
B2 ... 3621
Sigma1 ... 3622
Sigma2 ... 3622
GetValue ... 3623
CNormalMembershipFunction ... 3624
B ... 3626
Sigma ... 3626
GetValue ... 3627
CP_ShapedMembershipFunction ... 3628
A ... 3630
B ... 3630
C ... 3631
D ... 3631
GetValue ... 3631
CProductTwoSigmoidalMembershipFunctions ... 3633
A1 ... 3635
A2 ... 3635
C1 ... 3636
C2 ... 3636
GetValue ... 3637
CS_ShapedMembershipFunction ... 3638
A ... 3640
B ... 3640
GetValue ... 3641
CSigmoidalMembershipFunction ... 3642
A ... 3644
C ... 3644
GetValue ... 3645
CTrapezoidMembershipFunction ... 3646
X1 ... 3648
X2 ... 3648
X3 ... 3649
X4 ... 3649
GetValue ... 3650
CTriangularMembershipFunction ... 3651
X1 ... 3653
X2 ... 3653
X3 ... 3654
ToNormalMF ... 3654
GetValue ... 3654
CZ_ShapedMembershipFunction ... 3656
A ... 3658

        B ... 3658
        GetValue ... 3659
        IMembershipFunction ... 3660
        GetValue ... 3660
    Fuzzy systems rules ... 3661
        CMamdaniFuzzyRule ... 3662
        Conclusion ... 3662
        Weight ... 3663
        CSugenoFuzzyRule ... 3664
        Conclusion ... 3664
        CSingleCondition ... 3666
        Not ... 3666
        Term ... 3667
        Var ... 3667
        CConditions ... 3669
        ConditionsList ... 3669
        Not ... 3670
        Op ... 3670
        CGenericFuzzyRule ... 3671
        Conclusion ... 3671
        Condition ... 3672
        CreateCondition ... 3672
    Fuzzy systems variables ... 3674
        CFuzzyVariable ... 3675
        AddTerm ... 3676
        GetTermByName ... 3676
        Max ... 3676
        Min ... 3677
        Terms ... 3677
        Values ... 3677
        CSugenoVariable ... 3679
        Functions ... 3679
        GetFuncByName ... 3680
        Values ... 3680
    Fuzzy terms ... 3681
        MembershipFunction ... 3682
    Fuzzy systems ... 3683
        Mamdani system ... 3684
        AggregationMethod ... 3684
        Calculate ... 3685
        DefuzzificationMethod ... 3685
        EmptyRule ... 3685
        ImplicationMethod ... 3685
        Output ... 3686
        OutputByName ... 3686
        ParseRule ... 3686
        Rules ... 3686
        Sugeno system ... 3688
        Calculate ... 3688
        CreateSugenoFunction ... 3689
        EmptyRule ... 3690
        Output ... 3690
        OutputByName ... 3690
        ParseRule ... 3690
        Rules ... 3691
OpenCL ... 3692
    BufferCreate ... 3694
    BufferFree ... 3695

BufferFromArray ... 3696
BufferRead ... 3697
BufferWrite ... 3698
Execute ... 3699
GetContext ... 3700
GetKernel ... 3701
GetKernelName ... 3702
GetProgram ... 3703
Initialize ... 3704
KernelCreate ... 3705
KernelFree ... 3706
SetArgument ... 3707
SetArgumentBuffer ... 3708
SetArgumentLocalMemory ... 3709
SetBuffersCount ... 3710
SetKernelsCount ... 3711
Shutdown ... 3712
SupportDouble ... 3713
Basic Class CObject ... 3714
Prev ... 3715
Prev ... 3716
Next ... 3717
Next ... 3718
Compare ... 3719
Save ... 3721
Load ... 3723
Type ... 3725
Data Collections ... 3726
CArray ... 3727
Step ... 3729
Step ... 3730
Total ... 3731
Available ... 3732
Max ... 3733
IsSorted ... 3734
SortMode ... 3735
Clear ... 3736
Sort ... 3737
Save ... 3738
Load ... 3739
CArrayChar ... 3740
Reserve ... 3742
Resize ... 3743
Shutdown ... 3744
Add ... 3745
AddArray ... 3746
AddArray ... 3747
Insert ... 3749
InsertArray ... 3750
InsertArray ... 3751
AssignArray ... 3753
AssignArray ... 3754
Update ... 3756
Shift ... 3757
Delete ... 3758
DeleteRange ... 3759
At ... 3760
CompareArray ... 3762

CompareArray 3763
InsertSort 3764
Search 3765
SearchGreat 3766
SearchLess 3767
SearchGreatOrEqual 3768
SearchLessOrEqual 3769
SearchFirst 3770
SearchLast 3771
SearchLinear 3772
Save 3773
Load 3774
Type 3776
CArrayShort 3777
Reserve 3780
Resize 3781
Shutdown 3782
Add 3783
AddArray 3784
AddArray 3785
Insert 3787
InsertArray 3788
InsertArray 3789
AssignArray 3791
AssignArray 3792
Update 3794
Shift 3795
Delete 3796
DeleteRange 3797
At 3798
CompareArray 3800
CompareArray 3801
InsertSort 3802
Search 3803
SearchGreat 3804
SearchLess 3805
SearchGreatOrEqual 3806
SearchLessOrEqual 3807
SearchFirst 3808
SearchLast 3809
SearchLinear 3810
Save 3811
Load 3813
Type 3815
CArrayInt 3816
Reserve 3819
Resize 3820
Shutdown 3821
Add 3822
AddArray 3823
AddArray 3824
Insert 3826
InsertArray 3827
InsertArray 3828
AssignArray 3830
AssignArray 3831
Update 3833
Shift 3834

Delete ... 3835
DeleteRange ... 3836
At ... 3837
CompareArray ... 3839
CompareArray ... 3840
InsertSort ... 3841
Search ... 3842
SearchGreat ... 3843
SearchLess ... 3844
SearchGreatOrEqual ... 3845
SearchLessOrEqual ... 3846
SearchFirst ... 3847
SearchLast ... 3848
SearchLinear ... 3849
Save ... 3850
Load ... 3852
Type ... 3854
CArrayLong ... 3855
Reserve ... 3858
Resize ... 3859
Shutdown ... 3860
Add ... 3861
AddArray ... 3862
AddArray ... 3863
Insert ... 3865
InsertArray ... 3866
InsertArray ... 3867
AssignArray ... 3869
AssignArray ... 3870
Update ... 3872
Shift ... 3873
Delete ... 3874
DeleteRange ... 3875
At ... 3876
CompareArray ... 3878
CompareArray ... 3879
InsertSort ... 3880
Search ... 3881
SearchGreat ... 3882
SearchLess ... 3883
SearchGreatOrEqual ... 3884
SearchLessOrEqual ... 3885
SearchFirst ... 3886
SearchLast ... 3887
SearchLinear ... 3888
Save ... 3889
Load ... 3891
Type ... 3893
CArrayFloat ... 3894
Delta ... 3897
Reserve ... 3898
Resize ... 3899
Shutdown ... 3900
Add ... 3901
AddArray ... 3902
AddArray ... 3903
Insert ... 3905
InsertArray ... 3906

- InsertArray ... 3907
- AssignArray ... 3909
- AssignArray ... 3910
- Update ... 3912
- Shift ... 3913
- Delete ... 3914
- DeleteRange ... 3915
- At ... 3916
- CompareArray ... 3918
- CompareArray ... 3919
- InsertSort ... 3920
- Search ... 3921
- SearchGreat ... 3922
- SearchLess ... 3923
- SearchGreatOrEqual ... 3924
- SearchLessOrEqual ... 3925
- SearchFirst ... 3926
- SearchLast ... 3927
- SearchLinear ... 3928
- Save ... 3929
- Load ... 3931
- Type ... 3933

CArrayDouble ... 3934
- Delta ... 3937
- Reserve ... 3938
- Resize ... 3939
- Shutdown ... 3940
- Add ... 3941
- AddArray ... 3942
- AddArray ... 3943
- Insert ... 3945
- InsertArray ... 3946
- InsertArray ... 3947
- AssignArray ... 3949
- AssignArray ... 3950
- Update ... 3952
- Shift ... 3953
- Delete ... 3954
- DeleteRange ... 3955
- At ... 3956
- CompareArray ... 3958
- CompareArray ... 3959
- Minimum ... 3960
- Maximum ... 3961
- InsertSort ... 3962
- Search ... 3963
- SearchGreat ... 3964
- SearchLess ... 3965
- SearchGreatOrEqual ... 3966
- SearchLessOrEqual ... 3967
- SearchFirst ... 3968
- SearchLast ... 3969
- SearchLinear ... 3970
- Save ... 3971
- Load ... 3973
- Type ... 3975

CArrayString ... 3976
- Reserve ... 3979

Resize 3980
Shutdown 3981
Add 3982
AddArray 3983
AddArray 3984
Insert 3986
InsertArray 3987
InsertArray 3988
AssignArray 3990
AssignArray 3991
Update 3993
Shift 3994
Delete 3995
DeleteRange 3996
At 3997
CompareArray 3999
CompareArray 4000
InsertSort 4001
Search 4002
SearchGreat 4003
SearchLess 4004
SearchGreatOrEqual 4005
SearchLessOrEqual 4006
SearchFirst 4007
SearchLast 4008
SearchLinear 4009
Save 4010
Load 4012
Type 4014
CArrayObj 4015
FreeMode 4020
FreeMode 4021
Reserve 4023
Resize 4024
Clear 4025
Shutdown 4026
CreateElement 4027
Add 4029
AddArray 4030
Insert 4033
InsertArray 4035
AssignArray 4037
Update 4039
Shift 4040
Detach 4041
Delete 4042
DeleteRange 4043
At 4044
CompareArray 4045
InsertSort 4046
Search 4047
SearchGreat 4048
SearchLess 4049
SearchGreatOrEqual 4050
SearchLessOrEqual 4052
SearchFirst 4053
SearchLast 4054
Save 4055

Load 4056
Type 4058
CList 4059
FreeMode 4061
FreeMode 4062
Total 4064
IsSorted 4065
SortMode 4066
CreateElement 4067
Add 4068
Insert 4069
DetachCurrent 4071
DeleteCurrent 4072
Delete 4073
Clear 4074
IndexOf 4075
GetNodeAtIndex 4076
GetFirstNode 4077
GetPrevNode 4078
GetCurrentNode 4079
GetNextNode 4080
GetLastNode 4081
Sort 4082
MoveToIndex 4083
Exchange 4084
CompareList 4085
Search 4086
Save 4087
Load 4089
Type 4091
CTreeNode 4092
Owner 4097
Left 4098
Right 4099
Balance 4100
BalanceL 4101
BalanceR 4102
CreateSample 4103
RefreshBalance 4104
GetNext 4105
SaveNode 4106
LoadNode 4107
Type 4108
CTree 4109
Root 4115
CreateElement 4116
Insert 4117
Detach 4118
Delete 4119
Clear 4120
Find 4121
Save 4122
Load 4123
Type 4124
Generic Data Collections 4125
ICollection<T> 4127
Add 4128
Count 4129

- Contains 4130
- CopyTo 4131
- Clear 4132
- Remove 4133

IEqualityComparable<T> 4134
- Equals 4135
- HashCode 4136

IComparable<T> 4137
- Compare 4138

IComparer<T> 4139
- Compare 4140

IEqualityComparer<T> 4141
- Equals 4142
- HashCode 4143

IList<T> 4144
- TryGetValue 4145
- TrySetValue 4146
- Insert 4147
- IndexOf 4148
- LastIndexOf 4149
- RemoveAt 4150

IMap<TKey,TValue> 4151
- Add 4152
- Contains 4153
- Remove 4154
- TryGetValue 4155
- TrySetValue 4156
- CopyTo 4157

ISet<T> 4158
- ExceptWith 4160
- IntersectWith 4161
- SymmetricExceptWith 4162
- UnionWith 4163
- IsProperSubsetOf 4164
- IsProperSupersetOf 4165
- IsSubsetOf 4166
- IsSupersetOf 4167
- Overlaps 4168
- SetEquals 4169

CDefaultComparer<T> 4170
- Compare 4171

CDefaultEqualityComparer<T> 4172
- Equals 4173
- HashCode 4174

CRedBlackTreeNode<T> 4175
- Value 4176
- Parent 4177
- Left 4178
- Right 4179
- Color 4180
- IsLeaf 4181
- CreateEmptyNode 4182

CLinkedListNode<T> 4183
- List 4184
- Next 4185
- Previous 4186
- Value 4187

CKeyValuePair<TKey,TValue> 4188

- Key 4189
- Value 4190
- Clone 4191
- Compare 4192
- Equals 4193
- HashCode 4194

CArrayList<T> 4195
- Capacity 4197
- Count 4198
- Contains 4199
- TrimExcess 4200
- TryGetValue 4201
- TrySetValue 4202
- Add 4203
- AddRange 4204
- Insert 4205
- InsertRange 4206
- CopyTo 4207
- BinarySearch 4208
- IndexOf 4209
- LastIndexOf 4210
- Clear 4211
- Remove 4212
- RemoveAt 4213
- RemoveRange 4214
- Reverse 4215
- Sort 4216

CHashMap<TKey,TValue> 4217
- Add 4219
- Count 4220
- Comparer 4221
- Contains 4222
- ContainsKey 4223
- ContainsValue 4224
- CopyTo 4225
- Clear 4226
- Remove 4227
- TryGetValue 4228
- TrySetValue 4229

CHashSet<T> 4230
- Add 4232
- Count 4233
- Contains 4234
- Comparer 4235
- TrimExcess 4236
- CopyTo 4237
- Clear 4238
- Remove 4239
- ExceptWith 4240
- IntersectWith 4241
- SymmetricExceptWith 4242
- UnionWith 4243
- IsProperSubsetOf 4244
- IsProperSupersetOf 4245
- IsSubsetOf 4246
- IsSupersetOf 4247
- Overlaps 4248
- SetEquals 4249

CLinkedList<T> 4250
  Add 4252
  AddAfter 4253
  AddBefore 4254
  AddFirst 4255
  AddLast 4256
  Count 4257
  Head 4258
  First 4259
  Last 4260
  Contains 4261
  CopyTo 4262
  Clear 4263
  Remove 4264
  RemoveFirst 4265
  RemoveLast 4266
  Find 4267
  FindLast 4268
CQueue<T> 4269
  Add 4270
  Enqueue 4271
  Count 4272
  Contains 4273
  TrimExcess 4274
  CopyTo 4275
  Clear 4276
  Remove 4277
  Dequeue 4278
  Peek 4279
CRedBlackTree<T> 4280
  Add 4282
  Count 4283
  Root 4284
  Contains 4285
  Comparer 4286
  TryGetMin 4287
  TryGetMax 4288
  CopyTo 4289
  Clear 4290
  Remove 4291
  RemoveMin 4292
  RemoveMax 4293
  Find 4294
  FindMin 4295
  FindMax 4296
CSortedMap<TKey, TValue> 4297
  Add 4299
  Count 4300
  Comparer 4301
  Contains 4302
  ContainsKey 4303
  ContainsValue 4304
  CopyTo 4305
  Clear 4306
  Remove 4307
  TryGetValue 4308
  TrySetValue 4309
CSortedSet<T> 4310

    - Add 4312
    - Count 4313
    - Contains 4314
    - Comparer 4315
    - TryGetMin 4316
    - TryGetMax 4317
    - CopyTo 4318
    - Clear 4319
    - Remove 4320
    - ExceptWith 4321
    - IntersectWith 4322
    - SymmetricExceptWith 4323
    - UnionWith 4324
    - IsProperSubsetOf 4325
    - IsProperSupersetOf 4326
    - IsSubsetOf 4327
    - IsSupersetOf 4328
    - Overlaps 4329
    - SetEquals 4330
    - GetViewBetween 4331
    - GetReverse 4332
  - CStack<T> 4333
    - Add 4334
    - Count 4335
    - Contains 4336
    - TrimExcess 4337
    - CopyTo 4338
    - Clear 4339
    - Remove 4340
    - Push 4341
    - Peek 4342
    - Pop 4343
  - ArrayBinarySearch<T> 4344
  - ArrayIndexOf<T> 4345
  - ArrayLastIndexOf<T> 4346
  - ArrayReverse<T> 4347
  - Compare 4348
  - Equals<T> 4351
  - GetHashCode 4352
- **Files** **4355**
  - CFile 4356
    - Handle 4358
    - Filename 4359
    - Flags 4360
    - SetUnicode 4361
    - SetCommon 4362
    - Open 4363
    - Close 4364
    - Delete 4365
    - IsExist 4366
    - Copy 4367
    - Move 4368
    - Size 4369
    - Tell 4370
    - Seek 4371
    - Flush 4372
    - IsEnding 4373
    - IsLineEnding 4374

FolderCreate 4375
FolderDelete 4376
FolderClean 4377
FileFindFirst 4378
FileFindNext 4379
FileFindClose 4380
CFileBin 4381
Open 4383
WriteChar 4384
WriteShort 4385
WriteInteger 4386
WriteLong 4387
WriteFloat 4388
WriteDouble 4389
WriteString 4390
WriteCharArray 4391
WriteShortArray 4392
WriteIntegerArray 4393
WriteLongArray 4394
WriteFloatArray 4395
WriteDoubleArray 4396
WriteObject 4397
ReadChar 4398
ReadShort 4399
ReadInteger 4400
ReadLong 4401
ReadFloat 4402
ReadDouble 4403
ReadString 4404
ReadCharArray 4405
ReadShortArray 4406
ReadIntegerArray 4407
ReadLongArray 4408
ReadFloatArray 4409
ReadDoubleArray 4410
ReadObject 4411
CFileTxt 4412
Open 4413
WriteString 4414
ReadString 4415
Strings 4416
CString 4417
Str 4419
Len 4420
Copy 4421
Fill 4422
Assign 4423
Append 4424
Insert 4425
Compare 4426
CompareNoCase 4427
Left 4428
Right 4429
Mid 4430
Trim 4431
TrimLeft 4432
TrimRight 4433
Clear 4434

ToUpper 4435
ToLower 4436
Reverse 4437
Find 4438
FindRev 4439
Remove 4440
Replace 4441
Graphic Objects 4442
CChartObject 4443
ChartId 4446
Window 4447
Name 4448
NumPoints 4449
Attach 4450
SetPoint 4451
Delete 4452
Detach 4453
ShiftObject 4454
ShiftPoint 4455
Time 4456
Price 4458
Color 4460
Style 4461
Width 4462
Background 4463
Selected 4464
Selectable 4465
Description 4466
Tooltip 4467
Timeframes 4468
Z_Order 4469
CreateTime 4470
LevelsCount 4471
LevelColor 4472
LevelStyle 4474
LevelWidth 4476
LevelValue 4478
LevelDescription 4480
GetInteger 4482
SetInteger 4484
GetDouble 4486
SetDouble 4488
GetString 4490
SetString 4492
Save 4494
Load 4495
Type 4496
Line Objects 4497
CChartObjectVLine 4498
Create 4499
Type 4500
CChartObjectHLine 4501
Create 4502
Type 4503
CChartObjectTrend 4504
Create 4506
RayLeft 4507
RayRight 4508

- Save 4509
- Load 4510
- Type 4511
- CChartObjectTrendByAngle 4512
  - Create 4514
  - Angle 4515
  - Type 4516
- CChartObjectCycles 4517
  - Create 4518
  - Type 4519

Channel Objects 4520
- CChartObjectChannel 4521
  - Create 4523
  - Type 4524
- CChartObjectRegression 4525
  - Create 4527
  - Type 4528
- CChartObjectStdDevChannel 4529
  - Create 4531
  - Deviations 4532
  - Save 4533
  - Load 4534
  - Type 4535
- CChartObjectPitchfork 4536
  - Create 4538
  - Type 4539

Gann Tools 4540
- CChartObjectGannLine 4541
  - Create 4543
  - PipsPerBar 4544
  - Save 4545
  - Load 4546
  - Type 4547
- CChartObjectGannFan 4548
  - Create 4550
  - PipsPerBar 4551
  - Downtrend 4552
  - Save 4553
  - Load 4554
  - Type 4555
- CChartObjectGannGrid 4556
  - Create 4558
  - PipsPerBar 4559
  - Downtrend 4560
  - Save 4561
  - Load 4562
  - Type 4563

Fibonacci Tools 4564
- CChartObjectFibo 4565
  - Create 4567
  - Type 4568
- CChartObjectFiboTimes 4569
  - Create 4570
  - Type 4571
- CChartObjectFiboFan 4572
  - Create 4573
  - Type 4574
- CChartObjectFiboArc 4575

        Create 4577
        Scale 4578
        Ellipse 4579
        Save 4580
        Load 4581
        Type 4582
    CChartObjectFiboChannel 4583
        Create 4585
        Type 4586
    CChartObjectFiboExpansion 4587
        Create 4589
        Type 4590
Elliott Tools 4591
    CChartObjectElliottWave3 4592
        Create 4594
        Degree 4595
        Lines 4596
        Save 4597
        Load 4598
        Type 4599
    CChartObjectElliottWave5 4600
        Create 4602
        Type 4604
Shape Objects 4605
    CChartObjectRectangle 4606
        Create 4607
        Type 4608
    CChartObjectTriangle 4609
        Create 4610
        Type 4611
    CChartObjectEllipse 4612
        Create 4613
        Type 4614
Arrow Objects 4615
    CChartObjectArrow 4616
        Create 4618
        ArrowCode 4620
        Anchor 4622
        Save 4624
        Load 4625
        Type 4626
    Arrows with fixed code 4627
        Create 4629
        ArrowCode 4631
        Type 4632
Control Objects 4633
    CChartObjectText 4634
        Create 4636
        Angle 4637
        Font 4638
        FontSize 4639
        Anchor 4640
        Save 4641
        Load 4642
        Type 4643
    CChartObjectLabel 4644
        Create 4646
        X_Distance 4647

Y_Distance ... 4648
X_Size ... 4649
Y_Size ... 4650
Corner ... 4651
Time ... 4652
Price ... 4653
Save ... 4654
Load ... 4655
Type ... 4656
CChartObjectEdit ... 4657
Create ... 4659
TextAlign ... 4660
X_Size ... 4661
Y_Size ... 4662
BackColor ... 4663
BorderColor ... 4664
ReadOnly ... 4665
Angle ... 4666
Save ... 4667
Load ... 4668
Type ... 4669
CChartObjectButton ... 4670
State ... 4672
Save ... 4673
Load ... 4674
Type ... 4675
CChartObjectSubChart ... 4676
Create ... 4678
X_Distance ... 4679
Y_Distance ... 4680
Corner ... 4681
X_Size ... 4682
Y_Size ... 4683
Symbol ... 4684
Period ... 4685
Scale ... 4686
DateScale ... 4687
PriceScale ... 4688
Time ... 4689
Price ... 4690
Save ... 4691
Load ... 4692
Type ... 4693
CChartObjectBitmap ... 4694
Create ... 4696
BmpFile ... 4697
X_Offset ... 4698
Y_Offset ... 4699
Save ... 4700
Load ... 4701
Type ... 4702
CChartObjectBmpLabel ... 4703
Create ... 4705
X_Distance ... 4706
Y_Distance ... 4707
X_Offset ... 4708
Y_Offset ... 4709
Corner ... 4710

X_Size 4711
Y_Size 4712
BmpFileOn 4713
BmpFileOff 4714
State 4715
Time 4716
Price 4717
Save 4718
Load 4719
Type 4720
CChartObjectRectLabel 4721
Create 4723
X_Size 4724
Y_Size 4725
BackColor 4726
Angle 4727
BorderType 4728
Save 4729
Load 4730
Type 4731
Custom Graphics 4732
CCanvas 4733
Attach 4737
Arc 4738
Pie 4742
FillPolygon 4746
FillEllipse 4747
GetDefaultColor 4748
ChartObjectName 4749
Circle 4750
CircleAA 4751
CircleWu 4752
Create 4753
CreateBitmap 4754
CreateBitmapLabel 4756
Destroy 4758
Ellipse 4759
EllipseAA 4760
EllipseWu 4761
Erase 4762
Fill 4763
FillCircle 4764
FillRectangle 4765
FillTriangle 4766
FontAngleGet 4767
FontAngleSet 4768
FontFlagsGet 4769
FontFlagsSet 4770
FontGet 4771
FontNameGet 4772
FontNameSet 4773
FontSet 4774
FontSizeGet 4775
FontSizeSet 4776
Height 4777
Line 4778
LineAA 4779
LineWu 4780

LineHorizontal ... 4781
LineVertical ... 4782
LineStyleSet ... 4783
LineThick ... 4784
LineThickVertical ... 4785
LineThickHorizontal ... 4786
LoadFromFile ... 4787
PixelGet ... 4788
PixelSet ... 4789
PixelSetAA ... 4790
Polygon ... 4791
PolygonAA ... 4792
PolygonWu ... 4793
PolygonThick ... 4794
PolygonSmooth ... 4795
Polyline ... 4796
PolylineSmooth ... 4797
PolylineThick ... 4798
PolylineWu ... 4799
PolylineAA ... 4800
Rectangle ... 4801
Resize ... 4802
ResourceName ... 4803
TextHeight ... 4804
TextOut ... 4805
TextSize ... 4806
TextWidth ... 4807
TransparentLevelSet ... 4808
Triangle ... 4809
TriangleAA ... 4810
TriangleWu ... 4811
Update ... 4812
Width ... 4813
CChartCanvas ... 4814
ColorBackground ... 4818
ColorBorder ... 4819
ColorText ... 4820
ColorGrid ... 4821
MaxData ... 4822
MaxDescrLen ... 4823
ShowFlags ... 4824
IsShowLegend ... 4825
IsShowScaleLeft ... 4826
IsShowScaleRight ... 4827
IsShowScaleTop ... 4828
IsShowScaleBottom ... 4829
IsShowGrid ... 4830
IsShowDescriptors ... 4831
IsShowPercent ... 4832
VScaleMin ... 4833
VScaleMax ... 4834
NumGrid ... 4835
DataOffset ... 4836
DataTotal ... 4837
DrawDescriptors ... 4838
DrawData ... 4839
Create ... 4840
AllowedShowFlags ... 4841

  - ShowLegend ... 4842
  - ShowScaleLeft ... 4843
  - ShowScaleRight ... 4844
  - ShowScaleTop ... 4845
  - ShowScaleBottom ... 4846
  - ShowGrid ... 4847
  - ShowDescriptors ... 4848
  - ShowValue ... 4849
  - ShowPercent ... 4850
  - LegendAlignment ... 4851
  - Accumulative ... 4852
  - VScaleParams ... 4853
  - DescriptorUpdate ... 4854
  - ColorUpdate ... 4855
  - ValuesCheck ... 4856
  - Redraw ... 4857
  - DrawBackground ... 4858
  - DrawLegend ... 4859
  - DrawLegendVertical ... 4860
  - DrawLegendHorizontal ... 4861
  - CalcScales ... 4862
  - DrawScales ... 4863
  - DrawScaleLeft ... 4864
  - DrawScaleRight ... 4865
  - DrawScaleTop ... 4866
  - DrawScaleBottom ... 4867
  - DrawGrid ... 4868
  - DrawChart ... 4869
- CHistogramChart ... 4870
  - Gradient ... 4875
  - BarGap ... 4876
  - BarMinSize ... 4877
  - BarBorder ... 4878
  - Create ... 4879
  - SeriesAdd ... 4880
  - SeriesInsert ... 4881
  - SeriesUpdate ... 4882
  - SeriesDelete ... 4883
  - ValueUpdate ... 4884
  - DrawData ... 4885
  - DrawBar ... 4886
  - GradientBrush ... 4887
- CLineChart ... 4888
  - Filled ... 4892
  - Create ... 4893
  - SeriesAdd ... 4894
  - SeriesInsert ... 4895
  - SeriesUpdate ... 4896
  - SeriesDelete ... 4897
  - ValueUpdate ... 4898
  - DrawChart ... 4899
  - DrawData ... 4900
  - CalcArea ... 4901
- CPieChart ... 4902
  - Create ... 4906
  - SeriesSet ... 4907
  - ValueAdd ... 4908
  - ValueInsert ... 4909

ValueUpdate ... 4910
ValueDelete ... 4911
DrawChart ... 4912
DrawPie ... 4913
LabelMake ... 4914
3D Graphics ... 4915
CCanvas3D ... 4916
AmbientColorGet ... 4918
AmbientColorSet ... 4919
Attach ... 4920
Create ... 4921
Destroy ... 4922
DXContext ... 4923
DXDispatcher ... 4924
InputScene ... 4925
LightColorGet ... 4926
LightColorSet ... 4927
LightDirectionGet ... 4928
LightDirectionSet ... 4929
ObjectAdd ... 4930
ProjectionMatrixGet ... 4931
ProjectionMatrixSet ... 4932
Render ... 4933
RenderBegin ... 4934
RenderEnd ... 4935
ViewMatrixGet ... 4936
ViewMatrixSet ... 4937
ViewPositionSet ... 4938
ViewRotationSet ... 4939
ViewTargetSet ... 4940
ViewUpDirectionSet ... 4941
Price Charts ... 4942
ChartID ... 4947
Mode ... 4948
Foreground ... 4949
Shift ... 4950
ShiftSize ... 4951
AutoScroll ... 4952
Scale ... 4953
ScaleFix ... 4954
ScaleFix_11 ... 4955
FixedMax ... 4956
FixedMin ... 4957
PointsPerBar ... 4958
ScalePPB ... 4959
ShowOHLC ... 4960
ShowLineBid ... 4961
ShowLineAsk ... 4962
ShowLastLine ... 4963
ShowPeriodSep ... 4964
ShowGrid ... 4965
ShowVolumes ... 4966
ShowObjectDescr ... 4967
ShowDateScale ... 4968
ShowPriceScale ... 4969
ColorBackground ... 4970
ColorForeground ... 4971
ColorGrid ... 4972

ColorBarUp 4973
ColorBarDown 4974
ColorCandleBull 4975
ColorCandleBear 4976
ColorChartLine 4977
ColorVolumes 4978
ColorLineBid 4979
ColorLineAsk 4980
ColorLineLast 4981
ColorStopLevels 4982
VisibleBars 4983
WindowsTotal 4984
WindowIsVisible 4985
WindowHandle 4986
FirstVisibleBar 4987
WidthInBars 4988
WidthInPixels 4989
HeightInPixels 4990
PriceMin 4991
PriceMax 4992
Attach 4993
FirstChart 4994
NextChart 4995
Open 4996
Detach 4997
Close 4998
BringToTop 4999
EventObjectCreate 5000
EventObjectDelete 5001
IndicatorAdd 5002
IndicatorDelete 5003
IndicatorsTotal 5004
IndicatorName 5005
Navigate 5006
Symbol 5007
Period 5008
Redraw 5009
GetInteger 5010
SetInteger 5011
GetDouble 5012
SetDouble 5013
GetString 5014
SetString 5015
SetSymbolPeriod 5016
ApplyTemplate 5017
ScreenShot 5018
WindowOnDropped 5019
PriceOnDropped 5020
TimeOnDropped 5021
XOnDropped 5022
YOnDropped 5023
Save 5024
Load 5025
Type 5026

Scientific Charts 5027

GraphPlot 5028
CAxis 5032
  AutoScale 5034

Min 5035
Max 5036
Step 5037
Name 5038
Color 5039
ValuesSize 5040
ValuesWidth 5041
ValuesFormat 5042
ValuesDateTimeMode 5043
ValuesFunctionFormat 5044
ValuesFunctionFormatCBData 5046
NameSize 5047
ZeroLever 5048
DefaultStep 5049
MaxLabels 5050
MinGrace 5051
MaxGrace 5052
SelectAxisScale 5053
CColorGenerator 5054
Next 5055
Reset 5056
CCurve 5057
Type 5059
Name 5060
Color 5061
XMax 5062
XMin 5063
YMax 5064
YMin 5065
Size 5066
PointsSize 5067
PointsFill 5068
PointsColor 5069
GetX 5070
GetY 5071
LinesStyle 5072
LinesIsSmooth 5073
LinesSmoothTension 5074
LinesSmoothStep 5075
LinesEndStyle 5076
LinesWidth 5077
HistogramWidth 5079
CustomPlotCBData 5080
CustomPlotFunction 5081
PointsType 5085
StepsDimension 5086
TrendLineCoefficients 5087
TrendLineColor 5088
TrendLineVisible 5089
Update 5091
Visible 5093
CGraphic 5094
Create 5097
Destroy 5098
Update 5099
ChartObjectName 5100
ResourceName 5101
XAxis 5102

YAxis 5103
GapSize 5104
BackgroundColor 5105
BackgroundMain 5106
BackgroundMainSize 5107
BackgroundMainColor 5108
BackgroundSub 5109
BackgroundSubSize 5110
BackgroundSubColor 5111
GridLineColor 5112
GridBackgroundColor 5113
GridCircleRadius 5114
GridCircleColor 5115
GridHasCircle 5116
GridAxisLineColor 5117
HistoryNameWidth 5118
HistoryNameSize 5119
HistorySymbolSize 5120
TextAdd 5121
LineAdd 5122
CurveAdd 5123
CurvePlot 5126
CurvePlotAll 5127
CurveGetByIndex 5128
CurveGetByName 5129
CurveRemoveByIndex 5130
CurveRemoveByName 5131
CurvesTotal 5132
MarksToAxisAdd 5133
MajorMarkSize 5134
FontSet 5135
FontGet 5136
Attach 5137
CalculateMaxMinValues 5138
Height 5139
IndentDown 5140
IndentLeft 5141
IndentRight 5142
IndentUp 5143
Redraw 5144
ResetParameters 5145
ScaleX 5146
ScaleY 5147
SetDefaultParameters 5148
Width 5149
Indicators 5150
Base classes 5151
CSpreadBuffer 5152
Size 5154
SetSymbolPeriod 5155
At 5156
Refresh 5157
RefreshCurrent 5158
CTimeBuffer 5159
Size 5161
SetSymbolPeriod 5162
At 5163
Refresh 5164

RefreshCurrent 5165
CTickVolumeBuffer 5166
Size 5168
SetSymbolPeriod 5169
At 5170
Refresh 5171
RefreshCurrent 5172
CRealVolumeBuffer 5173
Size 5175
SetSymbolPeriod 5176
At 5177
Refresh 5178
RefreshCurrent 5179
CDoubleBuffer 5180
Size 5182
SetSymbolPeriod 5183
At 5184
Refresh 5185
RefreshCurrent 5186
COpenBuffer 5187
Refresh 5188
RefreshCurrent 5189
CHighBuffer 5190
Refresh 5191
RefreshCurrent 5192
CLowBuffer 5193
Refresh 5194
RefreshCurrent 5195
CCloseBuffer 5196
Refresh 5197
RefreshCurrent 5198
CIndicatorBuffer 5199
Offset 5201
Name 5202
At 5203
Refresh 5204
RefreshCurrent 5205
CSeries 5206
Name 5208
BuffersTotal 5209
Timeframe 5210
Symbol 5211
Period 5212
RefreshCurrent 5213
BufferSize 5214
BufferResize 5215
Refresh 5216
PeriodDescription 5217
CPriceSeries 5218
BufferResize 5220
GetData 5221
Refresh 5222
MinIndex 5223
MinValue 5224
MaxIndex 5225
MaxValue 5226
CIndicator 5227
Handle 5229

- Status 5230
- FullRelease 5231
- Create 5232
- BufferResize 5233
- BarsCalculated 5234
- GetData 5235
- Refresh 5238
- Minimum 5239
- MinValue 5240
- Maximum 5241
- MaxValue 5242
- MethodDescription 5243
- PriceDescription 5244
- VolumeDescription 5245
- AddToChart 5246
- DeleteFromChart 5247

CIndicators 5248
  - Create 5249
  - Refresh 5250

Timeseries classes 5251
  - CiSpread 5252
    - Create 5254
    - BufferResize 5255
    - GetData 5256
    - Refresh 5258
  - CiTime 5259
    - Create 5261
    - BufferResize 5262
    - GetData 5263
    - Refresh 5265
  - CiTickVolume 5266
    - Create 5268
    - BufferResize 5269
    - GetData 5270
    - Refresh 5272
  - CiRealVolume 5273
    - Create 5275
    - BufferResize 5276
    - GetData 5277
    - Refresh 5279
  - CiOpen 5280
    - Create 5282
    - GetData 5283
  - CiHigh 5285
    - Create 5287
    - GetData 5288
  - CiLow 5290
    - Create 5292
    - GetData 5293
  - CiClose 5295
    - Create 5297
    - GetData 5298

Trend Indicators 5300
  - CiADX 5301
    - MaPeriod 5303
    - Create 5304
    - Main 5305
    - Plus 5306

- Minus 5307
- Type 5308

CiADXWilder 5309

- MaPeriod 5311
- Create 5312
- Main 5313
- Plus 5314
- Minus 5315
- Type 5316

CiBands 5317

- MaPeriod 5319
- MaShift 5320
- Deviation 5321
- Applied 5322
- Create 5323
- Base 5324
- Upper 5325
- Lower 5326
- Type 5327

CiEnvelopes 5328

- MaPeriod 5330
- MaShift 5331
- MaMethod 5332
- Deviation 5333
- Applied 5334
- Create 5335
- Upper 5336
- Lower 5337
- Type 5338

CiIchimoku 5339

- TenkanSenPeriod 5341
- KijunSenPeriod 5342
- SenkouSpanBPeriod 5343
- Create 5344
- TenkanSen 5345
- KijunSen 5346
- SenkouSpanA 5347
- SenkouSpanB 5348
- ChinkouSpan 5349
- Type 5350

CiMA 5351

- MaPeriod 5353
- MaShift 5354
- MaMethod 5355
- Applied 5356
- Create 5357
- Main 5358
- Type 5359

CiSAR 5360

- SarStep 5362
- Maximum 5363
- Create 5364
- Main 5365
- Type 5366

CiStdDev 5367

- MaPeriod 5369
- MaShift 5370
- MaMethod 5371

    - Applied 5372
    - Create 5373
    - Main 5374
    - Type 5375
  - CiDEMA 5376
    - MaPeriod 5378
    - IndShift 5379
    - Applied 5380
    - Create 5381
    - Main 5382
    - Type 5383
  - CiTEMA 5384
    - MaPeriod 5386
    - IndShift 5387
    - Applied 5388
    - Create 5389
    - Main 5390
    - Type 5391
  - CiFrAMA 5392
    - MaPeriod 5394
    - IndShift 5395
    - Applied 5396
    - Create 5397
    - Main 5398
    - Type 5399
  - CiAMA 5400
    - MaPeriod 5402
    - FastEmaPeriod 5403
    - SlowEmaPeriod 5404
    - IndShift 5405
    - Applied 5406
    - Create 5407
    - Main 5408
    - Type 5409
  - CiVIDyA 5410
    - CmoPeriod 5412
    - EmaPeriod 5413
    - IndShift 5414
    - Applied 5415
    - Create 5416
    - Main 5417
    - Type 5418
- Oscillators 5419
  - CiATR 5420
    - MaPeriod 5422
    - Create 5423
    - Main 5424
    - Type 5425
  - CiBearsPower 5426
    - MaPeriod 5428
    - Create 5429
    - Main 5430
    - Type 5431
  - CiBullsPower 5432
    - MaPeriod 5434
    - Create 5435
    - Main 5436
    - Type 5437

CiCCI 5438
  MaPeriod 5440
  Applied 5441
  Create 5442
  Main 5443
  Type 5444
CiChaikin 5445
  FastMaPeriod 5447
  SlowMaPeriod 5448
  MaMethod 5449
  Applied 5450
  Create 5451
  Main 5452
  Type 5453
CiDeMarker 5454
  MaPeriod 5456
  Create 5457
  Main 5458
  Type 5459
CiForce 5460
  MaPeriod 5462
  MaMethod 5463
  Applied 5464
  Create 5465
  Main 5466
  Type 5467
CiMACD 5468
  FastEmaPeriod 5470
  SlowEmaPeriod 5471
  SignalPeriod 5472
  Applied 5473
  Create 5474
  Main 5475
  Signal 5476
  Type 5477
CiMomentum 5478
  MaPeriod 5480
  Applied 5481
  Create 5482
  Main 5483
  Type 5484
CiOsMA 5485
  FastEmaPeriod 5487
  SlowEmaPeriod 5488
  SignalPeriod 5489
  Applied 5490
  Create 5491
  Main 5492
  Type 5493
CiRSI 5494
  MaPeriod 5496
  Applied 5497
  Create 5498
  Main 5499
  Type 5500
CiRVI 5501
  MaPeriod 5503
  Create 5504

- Main 5505
- Signal 5506
- Type 5507
- CiStochastic 5508
  - Kperiod 5510
  - Dperiod 5511
  - Slowing 5512
  - MaMethod 5513
  - PriceField 5514
  - Create 5515
  - Main 5516
  - Signal 5517
  - Type 5518
- CiTriX 5519
  - MaPeriod 5521
  - Applied 5522
  - Create 5523
  - Main 5524
  - Type 5525
- CiWPR 5526
  - CalcPeriod 5528
  - Create 5529
  - Main 5530
  - Type 5531
- Volume Indicators 5532
  - CiAD 5533
    - Applied 5535
    - Create 5536
    - Main 5537
    - Type 5538
  - CiMFI 5539
    - MaPeriod 5541
    - Applied 5542
    - Create 5543
    - Main 5544
    - Type 5545
  - CiOBV 5546
    - Applied 5548
    - Create 5549
    - Main 5550
    - Type 5551
  - CiVolumes 5552
    - Applied 5554
    - Create 5555
    - Main 5556
    - Type 5557
- Bill Williams Indicators 5558
  - CiAC 5559
    - Create 5561
    - Main 5562
    - Type 5563
  - CiAlligator 5564
    - JawPeriod 5566
    - JawShift 5567
    - TeethPeriod 5568
    - TeethShift 5569
    - LipsPeriod 5570
    - LipsShift 5571

        MaMethod 5572
        Applied 5573
        Create 5574
        Jaw 5575
        Teeth 5576
        Lips 5577
        Type 5578
    CiAO 5579
        Create 5581
        Main 5582
        Type 5583
    CiFractals 5584
        Create 5586
        Upper 5587
        Lower 5588
        Type 5589
    CiGator 5590
        JawPeriod 5592
        JawShift 5593
        TeethPeriod 5594
        TeethShift 5595
        LipsPeriod 5596
        LipsShift 5597
        MaMethod 5598
        Applied 5599
        Create 5600
        Upper 5601
        Lower 5602
        Type 5603
    CiBWMFI 5604
        Applied 5606
        Create 5607
        Main 5608
        Type 5609
Custom indicators 5610
    NumBuffers 5611
    NumParams 5612
    ParamType 5613
    ParamLong 5614
    ParamDouble 5615
    ParamString 5616
    Type 5617
Trade Classes 5618
CAccountInfo 5619
    Login 5621
    TradeMode 5622
    TradeModeDescription 5623
    Leverage 5624
    StopoutMode 5625
    StopoutModeDescription 5626
    MarginMode 5627
    MarginModeDescription 5628
    TradeAllowed 5629
    TradeExpert 5630
    LimitOrders 5631
    Balance 5632
    Credit 5633
    Profit 5634

Equity 5635
Margin 5636
FreeMargin 5637
MarginLevel 5638
MarginCall 5639
MarginStopOut 5640
Name 5641
Server 5642
Currency 5643
Company 5644
InfoInteger 5645
InfoDouble 5646
InfoString 5647
OrderProfitCheck 5648
MarginCheck 5649
FreeMarginCheck 5650
MaxLotCheck 5651
CSymbolInfo 5652
Refresh 5656
RefreshRates 5657
Name 5658
Select 5659
IsSynchronized 5660
Volume 5661
VolumeHigh 5662
VolumeLow 5663
Time 5664
Spread 5665
SpreadFloat 5666
TicksBookDepth 5667
StopsLevel 5668
FreezeLevel 5669
Bid 5670
BidHigh 5671
BidLow 5672
Ask 5673
AskHigh 5674
AskLow 5675
Last 5676
LastHigh 5677
LastLow 5678
TradeCalcMode 5679
TradeCalcModeDescription 5680
TradeMode 5681
TradeModeDescription 5682
TradeExecution 5683
TradeExecutionDescription 5684
SwapMode 5685
SwapModeDescription 5686
SwapRollover3days 5687
SwapRollover3daysDescription 5688
MarginInitial 5689
MarginMaintenance 5690
MarginLong 5691
MarginShort 5692
MarginLimit 5693
MarginStop 5694
MarginStopLimit 5695

TradeTimeFlags ... 5696
TradeFillFlags ... 5697
Digits ... 5698
Point ... 5699
TickValue ... 5700
TickValueProfit ... 5701
TickValueLoss ... 5702
TickSize ... 5703
ContractSize ... 5704
LotsMin ... 5705
LotsMax ... 5706
LotsStep ... 5707
LotsLimit ... 5708
SwapLong ... 5709
SwapShort ... 5710
CurrencyBase ... 5711
CurrencyProfit ... 5712
CurrencyMargin ... 5713
Bank ... 5714
Description ... 5715
Path ... 5716
SessionDeals ... 5717
SessionBuyOrders ... 5718
SessionSellOrders ... 5719
SessionTurnover ... 5720
SessionInterest ... 5721
SessionBuyOrdersVolume ... 5722
SessionSellOrdersVolume ... 5723
SessionOpen ... 5724
SessionClose ... 5725
SessionAW ... 5726
SessionPriceSettlement ... 5727
SessionPriceLimitMin ... 5728
SessionPriceLimitMax ... 5729
InfoInteger ... 5730
InfoDouble ... 5731
InfoString ... 5732
NormalizePrice ... 5733
COrderInfo ... 5734
Ticket ... 5736
TimeSetup ... 5737
TimeSetupMsc ... 5738
OrderType ... 5739
TypeDescription ... 5740
State ... 5741
StateDescription ... 5742
TimeExpiration ... 5743
TimeDone ... 5744
TimeDoneMsc ... 5745
TypeFilling ... 5746
TypeFillingDescription ... 5747
TypeTime ... 5748
TypeTimeDescription ... 5749
Magic ... 5750
PositionId ... 5751
VolumeInitial ... 5752
VolumeCurrent ... 5753
PriceOpen ... 5754

StopLoss 5755
TakeProfit 5756
PriceCurrent 5757
PriceStopLimit 5758
Symbol 5759
Comment 5760
InfoInteger 5761
InfoDouble 5762
InfoString 5763
StoreState 5764
CheckState 5765
Select 5766
SelectByIndex 5767
CHistoryOrderInfo 5768
TimeSetup 5770
TimeSetupMsc 5771
OrderType 5772
TypeDescription 5773
State 5774
StateDescription 5775
TimeExpiration 5776
TimeDone 5777
TimeDoneMsc 5778
TypeFilling 5779
TypeFillingDescription 5780
TypeTime 5781
TypeTimeDescription 5782
Magic 5783
PositionId 5784
VolumeInitial 5785
VolumeCurrent 5786
PriceOpen 5787
StopLoss 5788
TakeProfit 5789
PriceCurrent 5790
PriceStopLimit 5791
Symbol 5792
Comment 5793
InfoInteger 5794
InfoDouble 5795
InfoString 5796
Ticket 5797
SelectByIndex 5798
CPositionInfo 5799
Time 5801
TimeMsc 5802
TimeUpdate 5803
TimeUpdateMsc 5804
PositionType 5805
TypeDescription 5806
Magic 5807
Identifier 5808
Volume 5809
PriceOpen 5810
StopLoss 5811
TakeProfit 5812
PriceCurrent 5813
Commission 5814

- Swap 5815
- Profit 5816
- Symbol 5817
- Comment 5818
- InfoInteger 5819
- InfoDouble 5820
- InfoString 5821
- Select 5822
- SelectByIndex 5823
- SelectByMagic 5824
- SelectByTicket 5825
- StoreState 5826
- CheckState 5827

CDealInfo 5828

- Order 5830
- Time 5831
- TimeMsc 5832
- DealType 5833
- TypeDescription 5834
- Entry 5835
- EntryDescription 5836
- Magic 5837
- PositionId 5838
- Volume 5839
- Price 5840
- Commision 5841
- Swap 5842
- Profit 5843
- Symbol 5844
- Comment 5845
- InfoInteger 5846
- InfoDouble 5847
- InfoString 5848
- Ticket 5849
- SelectByIndex 5850

CTrade 5851

- LogLevel 5855
- SetExpertMagicNumber 5856
- SetDeviationInPoints 5857
- SetTypeFilling 5858
- SetTypeFillingBySymbol 5859
- SetAsyncMode 5860
- SetMarginMode 5861
- OrderOpen 5862
- OrderModify 5864
- OrderDelete 5865
- PositionOpen 5866
- PositionModify 5867
- PositionClose 5868
- PositionClosePartial 5869
- PositionCloseBy 5871
- Buy 5872
- Sell 5873
- BuyLimit 5874
- BuyStop 5875
- SellLimit 5876
- SellStop 5877
- Request 5878

RequestAction 5879
RequestActionDescription 5880
RequestMagic 5881
RequestOrder 5882
RequestSymbol 5883
RequestVolume 5884
RequestPrice 5885
RequestStopLimit 5886
RequestSL 5887
RequestTP 5888
RequestDeviation 5889
RequestType 5890
RequestTypeDescription 5891
RequestTypeFilling 5892
RequestTypeFillingDescription 5893
RequestTypeTime 5894
RequestTypeTimeDescription 5895
RequestExpiration 5896
RequestComment 5897
RequestPosition 5898
RequestPositionBy 5899
Result 5900
ResultRetcode 5901
ResultRetcodeDescription 5902
ResultDeal 5903
ResultOrder 5904
ResultVolume 5905
ResultPrice 5906
ResultBid 5907
ResultAsk 5908
ResultComment 5909
CheckResult 5910
CheckResultRetcode 5911
CheckResultRetcodeDescription 5912
CheckResultBalance 5913
CheckResultEquity 5914
CheckResultProfit 5915
CheckResultMargin 5916
CheckResultMarginFree 5917
CheckResultMarginLevel 5918
CheckResultComment 5919
PrintRequest 5920
PrintResult 5921
FormatRequest 5922
FormatRequestResult 5923
CTerminalInfo 5924
Build 5926
IsConnected 5927
IsDLLsAllowed 5928
IsTradeAllowed 5929
IsEmailEnabled 5930
IsFtpEnabled 5931
MaxBars 5932
CodePage 5933
CPUCores 5934
MemoryPhysical 5935
MemoryTotal 5936
MemoryAvailable 5937

- MemoryUsed ... 5938
- IsX64 ... 5939
- OpenCLSupport ... 5940
- DiskSpace ... 5941
- Language ... 5942
- Name ... 5943
- Company ... 5944
- Path ... 5945
- DataPath ... 5946
- CommonDataPath ... 5947
- InfoInteger ... 5948
- InfoString ... 5949

Strategy Modules ... 5950

- Base classes for Expert Advisors ... 5953
  - CExpertBase ... 5954
    - InitPhase ... 5956
    - TrendType ... 5957
    - UsedSeries ... 5958
    - EveryTick ... 5959
    - Open ... 5960
    - High ... 5961
    - Low ... 5962
    - Close ... 5963
    - Spread ... 5964
    - Time ... 5965
    - TickVolume ... 5966
    - RealVolume ... 5967
    - Init ... 5968
    - Symbol ... 5969
    - Period ... 5970
    - Magic ... 5971
    - ValidationSettings ... 5972
    - SetPriceSeries ... 5973
    - SetOtherSeries ... 5974
    - InitIndicators ... 5975
    - InitOpen ... 5976
    - InitHigh ... 5977
    - InitLow ... 5978
    - InitClose ... 5979
    - InitSpread ... 5980
    - InitTime ... 5981
    - InitTickVolume ... 5982
    - InitRealVolume ... 5983
    - PriceLevelUnit ... 5984
    - StartIndex ... 5985
    - CompareMagic ... 5986
  - CExpert ... 5987
    - Init ... 5992
    - Magic ... 5993
    - InitSignal ... 5994
    - InitTrailing ... 5995
    - InitMoney ... 5996
    - InitTrade ... 5997
    - Deinit ... 5998
    - OnTickProcess ... 5999
    - OnTradeProcess ... 6000
    - OnTimerProcess ... 6001
    - OnChartEventProcess ... 6002

OnBookEventProcess 6003
MaxOrders 6004
Signal 6005
ValidationSettings 6006
InitIndicators 6007
OnTick 6008
OnTrade 6009
OnTimer 6010
OnChartEvent 6011
OnBookEvent 6012
InitParameters 6013
DeinitTrade 6014
DeinitSignal 6015
DeinitTrailing 6016
DeinitMoney 6017
DeinitIndicators 6018
Refresh 6019
Processing 6020
SelectPosition 6022
CheckOpen 6023
CheckOpenLong 6024
CheckOpenShort 6025
OpenLong 6026
OpenShort 6027
CheckReverse 6028
CheckReverseLong 6029
CheckReverseShort 6030
ReverseLong 6031
ReverseShort 6033
CheckClose 6035
CheckCloseLong 6036
CheckCloseShort 6037
CloseAll 6038
Close 6039
CloseLong 6040
CloseShort 6041
CheckTrailingStop 6042
CheckTrailingStopLong 6043
CheckTrailingStopShort 6044
TrailingStopLong 6045
TrailingStopShort 6046
CheckTrailingOrderLong 6047
CheckTrailingOrderShort 6048
TrailingOrderLong 6049
TrailingOrderShort 6050
CheckDeleteOrderLong 6051
CheckDeleteOrderShort 6052
DeleteOrders 6053
DeleteOrder 6054
DeleteOrderLong 6055
DeleteOrderShort 6056
LotOpenLong 6057
LotOpenShort 6058
LotReverse 6059
PrepareHistoryDate 6060
HistoryPoint 6061
CheckTradeState 6062
WaitEvent 6063

NoWaitEvent ... 6064
TradeEventPositionStopTake ... 6065
TradeEventOrderTriggered ... 6066
TradeEventPositionOpened ... 6067
TradeEventPositionVolumeChanged ... 6068
TradeEventPositionModified ... 6069
TradeEventPositionClosed ... 6070
TradeEventOrderPlaced ... 6071
TradeEventOrderModified ... 6072
TradeEventOrderDeleted ... 6073
TradeEventNotIdentified ... 6074
TimeframeAdd ... 6075
TimeframesFlags ... 6076
CExpertSignal ... 6077
BasePrice ... 6080
UsedSeries ... 6081
Weight ... 6082
PatternsUsage ... 6083
General ... 6084
Ignore ... 6085
Invert ... 6086
ThresholdOpen ... 6087
ThresholdClose ... 6088
PriceLevel ... 6089
StopLevel ... 6090
TakeLevel ... 6091
Expiration ... 6092
Magic ... 6093
ValidationSettings ... 6094
InitIndicators ... 6095
AddFilter ... 6096
CheckOpenLong ... 6097
CheckOpenShort ... 6098
OpenLongParams ... 6099
OpenShortParams ... 6100
CheckCloseLong ... 6101
CheckCloseShort ... 6102
CloseLongParams ... 6103
CloseShortParams ... 6104
CheckReverseLong ... 6105
CheckReverseShort ... 6106
CheckTrailingOrderLong ... 6107
CheckTrailingOrderShort ... 6108
LongCondition ... 6109
ShortCondition ... 6110
Direction ... 6111
CExpertTrailing ... 6112
CheckTrailingStopLong ... 6114
CheckTrailingStopShort ... 6115
CExpertMoney ... 6116
Percent ... 6117
ValidationSettings ... 6118
CheckOpenLong ... 6119
CheckOpenShort ... 6120
CheckReverse ... 6121
CheckClose ... 6122
Modules of Trade Signals ... 6123
Signals of the Indicator Accelerator Oscillator ... 6126

Signals of the Indicator Adaptive Moving Average 6129
Signals of the Indicator Awesome Oscillator 6133
Signals of the Oscillator Bears Power 6137
Signals of the Oscillator Bulls Power 6139
Signals of the Oscillator Commodity Channel Index 6141
Signals of the Oscillator DeMarker 6145
Signals of the Indicator Double Exponential Moving Average 6149
Signals of the Indicator Envelopes 6153
Signals of the Indicator Fractal Adaptive Moving Average 6156
Signals of the Intraday Time Filter 6160
Signals of the Oscillator MACD 6162
Signals of the Indicator Moving Average 6168
Signals of the Indicator Parabolic SAR 6172
Signals of the Oscillator Relative Strength Index 6174
Signals of the Oscillator Relative Vigor Index 6180
Signals of the Oscillator Stochastic 6182
Signals of the Oscillator Triple Exponential Average 6187
Signals of the Indicator Triple Exponential Moving Average 6191
Signals of the Oscillator Williams Percent Range 6195
Trailing Stop Classes 6198
CTrailingFixedPips 6199
StopLevel 6201
ProfitLevel 6202
ValidationSettings 6203
CheckTrailingStopLong 6204
CheckTrailingStopShort 6205
CTrailingMA 6206
Period 6208
Shift 6209
Method 6210
Applied 6211
InitIndicators 6212
ValidationSettings 6213
CheckTrailingStopLong 6214
CheckTrailingStopShort 6215
CTrailingNone 6216
CheckTrailingStopLong 6217
CheckTrailingStopShort 6218
CTrailingPSAR 6219
Step 6221
Maximum 6222
InitIndicators 6223
CheckTrailingStopLong 6224
CheckTrailingStopShort 6225
Money Management Classes 6226
CMoneyFixedLot 6227
Lots 6229
ValidationSettings 6230
CheckOpenLong 6231
CheckOpenShort 6232
CMoneyFixedMargin 6233
CheckOpenLong 6234
CheckOpenShort 6235
CMoneyFixedRisk 6236
CheckOpenLong 6237
CheckOpenShort 6238
CMoneyNone 6239
ValidationSettings 6240

- CheckOpenLong 6241
- CheckOpenShort 6242
- CMoneySizeOptimized 6243
  - DecreaseFactor 6245
  - ValidationSettings 6246
  - CheckOpenLong 6247
  - CheckOpenShort 6248

Panels and Dialogs 6249

- CRect 6251
  - Left 6252
  - Top 6253
  - Right 6254
  - Bottom 6255
  - Width 6256
  - Height 6257
  - SetBound 6258
  - Move 6259
  - Shift 6260
  - Contains 6261
  - Format 6262
- CDateTime 6263
  - MonthName 6265
  - ShortMonthName 6266
  - DayName 6267
  - ShortDayName 6268
  - DaysInMonth 6269
  - DateTime 6270
  - Date 6271
  - Time 6272
  - Sec 6273
  - Min 6274
  - Hour 6275
  - Day 6276
  - Mon 6277
  - Year 6278
  - SecDec 6279
  - SecInc 6280
  - MinDec 6281
  - MinInc 6282
  - HourDec 6283
  - HourInc 6284
  - DayDec 6285
  - DayInc 6286
  - MonDec 6287
  - MonInc 6288
  - YearDec 6289
  - YearInc 6290
- CWnd 6291
  - Create 6295
  - Destroy 6296
  - OnEvent 6297
  - OnMouseEvent 6298
  - Name 6299
  - ControlsTotal 6300
  - Control 6301
  - ControlFind 6302
  - Rect 6303
  - Left 6304

- Top ... 6305
- Right ... 6306
- Bottom ... 6307
- Width ... 6308
- Height ... 6309
- Move ... 6310
- Shift ... 6311
- Resize ... 6312
- Contains ... 6313
- Alignment ... 6314
- Align ... 6315
- Id ... 6316
- IsEnabled ... 6317
- Enable ... 6318
- Disable ... 6319
- IsVisible ... 6320
- Visible ... 6321
- Show ... 6322
- Hide ... 6323
- IsActive ... 6324
- Activate ... 6325
- Deactivate ... 6326
- StateFlags ... 6327
- StateFlagsSet ... 6328
- StateFlagsReset ... 6329
- PropFlags ... 6330
- PropFlagsSet ... 6331
- PropFlagsReset ... 6332
- MouseX ... 6333
- MouseY ... 6334
- MouseFlags ... 6335
- MouseFocusKill ... 6336
- OnCreate ... 6337
- OnDestroy ... 6338
- OnMove ... 6339
- OnResize ... 6340
- OnEnable ... 6341
- OnDisable ... 6342
- OnShow ... 6343
- OnHide ... 6344
- OnActivate ... 6345
- OnDeactivate ... 6346
- OnClick ... 6347
- OnChange ... 6348
- OnMouseDown ... 6349
- OnMouseUp ... 6350
- OnDragStart ... 6351
- OnDragProcess ... 6352
- OnDragEnd ... 6353
- DragObjectCreate ... 6354
- DragObjectDestroy ... 6355

CWndObj ... 6356
- OnEvent ... 6358
- Text ... 6359
- Color ... 6360
- ColorBackground ... 6361
- ColorBorder ... 6362
- Font ... 6363

FontSize 6364
ZOrder 6365
OnObjectCreate 6366
OnObjectChange 6367
OnObjectDelete 6368
OnObjectDrag 6369
OnSetText 6370
OnSetColor 6371
OnSetColorBackground 6372
OnSetFont 6373
OnSetFontSize 6374
OnSetZOrder 6375
OnDestroy 6376
OnChange 6377
CWndContainer 6378
Destroy 6380
OnEvent 6381
OnMouseEvent 6382
ControlsTotal 6383
Control 6384
ControlFind 6385
Add 6386
Delete 6387
Move 6388
Shift 6389
Id 6390
Enable 6391
Disable 6392
Show 6393
Hide 6394
MouseFocusKill 6395
Save 6396
Load 6397
OnResize 6398
OnActivate 6399
OnDeactivate 6400
CLabel 6401
Create 6406
OnSetText 6407
OnSetColor 6408
OnSetFont 6409
OnSetFontSize 6410
OnCreate 6411
OnShow 6412
OnHide 6413
OnMove 6414
CBmpButton 6415
Create 6422
Border 6423
BmpNames 6424
BmpOffName 6425
BmpOnName 6426
BmpPassiveName 6427
BmpActiveName 6428
Pressed 6429
Locking 6430
OnSetZOrder 6431
OnCreate 6432

- OnShow ... 6433
- OnHide ... 6434
- OnMove ... 6435
- OnChange ... 6436
- OnActivate ... 6437
- OnDeactivate ... 6438
- OnMouseDown ... 6439
- OnMouseUp ... 6440

CButton ... 6441
- Create ... 6448
- Pressed ... 6449
- Locking ... 6450
- OnSetText ... 6451
- OnSetColor ... 6452
- OnSetColorBackground ... 6453
- OnSetColorBorder ... 6454
- OnSetFont ... 6455
- OnSetFontSize ... 6456
- OnCreate ... 6457
- OnShow ... 6458
- OnHide ... 6459
- OnMove ... 6460
- OnResize ... 6461
- OnMouseDown ... 6462
- OnMouseUp ... 6463

CEdit ... 6464
- Create ... 6469
- ReadOnly ... 6470
- TextAlign ... 6471
- OnObjectEndEdit ... 6472
- OnSetText ... 6473
- OnSetColor ... 6474
- OnSetColorBackground ... 6475
- OnSetColorBorder ... 6476
- OnSetFont ... 6477
- OnSetFontSize ... 6478
- OnSetZOrder ... 6479
- OnCreate ... 6480
- OnShow ... 6481
- OnHide ... 6482
- OnMove ... 6483
- OnResize ... 6484
- OnChange ... 6485
- OnClick ... 6486

CPanel ... 6487
- Create ... 6492
- BorderType ... 6493
- OnSetText ... 6494
- OnSetColorBackground ... 6495
- OnSetColorBorder ... 6496
- OnCreate ... 6497
- OnShow ... 6498
- OnHide ... 6499
- OnMove ... 6500
- OnResize ... 6501
- OnChange ... 6502

CPicture ... 6503
- Create ... 6508

- Border ... 6509
- BmpName ... 6510
- OnCreate ... 6511
- OnShow ... 6512
- OnHide ... 6513
- OnMove ... 6514
- OnChange ... 6515

CScroll ... 6516
- Create ... 6518
- OnEvent ... 6519
- MinPos ... 6520
- MaxPos ... 6521
- CurrPos ... 6522
- CreateBack ... 6523
- CreateInc ... 6524
- CreateDec ... 6525
- CreateThumb ... 6526
- OnClickInc ... 6527
- OnClickDec ... 6528
- OnShow ... 6529
- OnHide ... 6530
- OnChangePos ... 6531
- OnThumbDragStart ... 6532
- OnThumbDragProcess ... 6533
- OnThumbDragEnd ... 6534
- CalcPos ... 6535

CScrollV ... 6536
- CreateInc ... 6542
- CreateDec ... 6543
- CreateThumb ... 6544
- OnResize ... 6545
- OnChangePos ... 6546
- OnThumbDragStart ... 6547
- OnThumbDragProcess ... 6548
- OnThumbDragEnd ... 6549
- CalcPos ... 6550

CScrollH ... 6551
- CreateInc ... 6557
- CreateDec ... 6558
- CreateThumb ... 6559
- OnResize ... 6560
- OnChangePos ... 6561
- OnThumbDragStart ... 6562
- OnThumbDragProcess ... 6563
- OnThumbDragEnd ... 6564
- CalcPos ... 6565

CWndClient ... 6566
- Create ... 6569
- OnEvent ... 6570
- ColorBackground ... 6571
- ColorBorder ... 6572
- BorderType ... 6573
- VScrolled ... 6574
- HScrolled ... 6575
- CreateBack ... 6576
- CreateScrollV ... 6577
- CreateScrollH ... 6578
- OnResize ... 6579

OnVScrollShow 6580
OnVScrollHide 6581
OnHScrollShow 6582
OnHScrollHide 6583
OnScrollLineDown 6584
OnScrollLineUp 6585
OnScrollLineLeft 6586
OnScrollLineRight 6587
Rebound 6588
CListView 6589
Create 6595
OnEvent 6596
TotalView 6597
AddItem 6598
Select 6599
SelectByText 6600
SelectByValue 6601
Value 6602
CreateRow 6603
OnResize 6604
OnVScrollShow 6605
OnVScrollHide 6606
OnScrollLineDown 6607
OnScrollLineUp 6608
OnItemClick 6609
Redraw 6610
RowState 6611
CheckView 6612
CComboBox 6613
Create 6619
OnEvent 6620
AddItem 6621
ListViewItems 6622
Select 6623
SelectByText 6624
SelectByValue 6625
Value 6626
CreateEdit 6627
CreateButton 6628
CreateList 6629
OnClickEdit 6630
OnClickButton 6631
OnChangeList 6632
ListShow 6633
ListHide 6634
CCheckBox 6635
Create 6641
OnEvent 6642
Text 6643
Color 6644
Checked 6645
Value 6646
CreateButton 6647
CreateLabel 6648
OnClickButton 6649
OnClickLabel 6650
CCheckGroup 6651
Create 6657

OnEvent 6658
AddItem 6659
Value 6660
CreateButton 6661
OnVScrollShow 6662
OnVScrollHide 6663
OnScrollLineDown 6664
OnScrollLineUp 6665
OnChangeItem 6666
Redraw 6667
RowState 6668
CRadioButton 6669
Create 6671
OnEvent 6672
Text 6673
Color 6674
State 6675
CreateButton 6676
CreateLabel 6677
OnClickButton 6678
OnClickLabel 6679
CRadioGroup 6680
Create 6686
OnEvent 6687
AddItem 6688
Value 6689
CreateButton 6690
OnVScrollShow 6691
OnVScrollHide 6692
OnScrollLineDown 6693
OnScrollLineUp 6694
OnChangeItem 6695
Redraw 6696
RowState 6697
Select 6698
CSpinEdit 6699
Create 6704
OnEvent 6705
MinValue 6706
MaxValue 6707
Value 6708
CreateEdit 6709
CreateInc 6710
CreateDec 6711
OnClickInc 6712
OnClickDec 6713
OnChangeValue 6714
CDialog 6715
Create 6717
OnEvent 6718
Caption 6719
Add 6720
CreateWhiteBorder 6721
CreateBackground 6722
CreateCaption 6723
CreateButtonClose 6724
CreateClientArea 6725
OnClickCaption 6726

OnClickButtonClose 6727
ClientAreaVisible 6728
ClientAreaLeft 6729
ClientAreaTop 6730
ClientAreaRight 6731
ClientAreaBottom 6732
ClientAreaWidth 6733
ClientAreaHeight 6734
OnDialogDragStart 6735
OnDialogDragProcess 6736
OnDialogDragEnd 6737
CAppDialog 6738
Create 6741
Destroy 6742
OnEvent 6743
Run 6744
ChartEvent 6745
Minimized 6746
IniFileSave 6747
IniFileLoad 6748
IniFileName 6749
IniFileExt 6750
CreateCommon 6751
CreateExpert 6752
CreateIndicator 6753
CreateButtonMinMax 6754
OnClickButtonClose 6755
OnClickButtonMinMax 6756
OnAnotherApplicationClose 6757
Rebound 6758
Minimize 6759
Maximize 6760
CreateInstanceId 6761
ProgramName 6762
SubwinOff 6763
36 Moving from MQL4 6764
37 List of MQL5 Functions 6767
38 List of MQL5 Constants 6801

MQL5 Reference

MetaQuotes Language 5 (MQL5) is a high-level language designed for developing technical indicators, trading robots and utility applications, which automate financial trading. MQL5 has been developed by MetaQuotes for their trading platform. The language syntax is very close to C++ enabling programmers to develop applications in the object-oriented programming (OOP) style.

In addition to the MQL5 language, the trading platform package also includes the MetaEditor IDE with highly advanced code writing tools, such as templates, snippets, debugging, profiling and auto completion tools, as well as built-in MQL5 Storage enabling file versioning.

The language support is available on the MQL5 Algotrading community website, which contains a huge free CodeBase and a plethora of articles. These articles cover all the aspects of the modern trading, including neural networks, statistics and analysis, high-frequency trading, arbitrage, testing and optimization of trading strategies, use of trading automation robots, and more.

Traders and MQL5 program developers can communicate on the forum, order and develop applications using the Freelance service, as well as buy and sell protected programs in the Market of automated trading applications.

The MQL5 language provides specialized trading functions and predefined event handlers to help programmers develop Expert Advisors (EAs), which automatically control trading processes following specific trading rules. In addition to EAs, MQL5 allows developing custom technical indicators, scripts and libraries.

This MQL5 language reference contains functions, operations, reserved words and other language constructions divided into categories. The reference also provides descriptions of Standard Library classes used for developing trading strategies, control panels, custom graphics and enabling file access.

Additionally, the CodeBase contains the ALGLIB numerical analysis library, which can be used for solving various mathematical problems.

Algo Trading Books

Starting to learn something new is always challenging. To assist beginners, we have released two comprehensive books on MQL5 programming, designed for anyone who wish to master the creation of trading robots and applications for algorithmic trading.

These books offer a systematic and structured presentation of the material to make the learning process significantly easier. Detailed code examples, which explain the step-by-step creation of trading robots and applications, allow for a deeper understanding of algorithmic trading nuances. The books include numerous practical exercises to help reinforce the acquired knowledge and develop programming skills in real trading environments.

"MQL5 Programming for Traders" is the most complete and detailed tutorial on MQL5, suitable for programmers of all levels. Beginners will learn the basics: the book introduces development tools and basic programming concepts. Based on this material, you will create, compile and run your first application in the MetaTrader 5 trading platform. Users with experience in other programming languages can immediately proceed to the application part: creating trading robots and analytical applications in MQL5.

"Neural Networks for Algorithmic Trading with MQL5" is a guide to using machine learning methods in trading robots for the MetaTrader 5 platform. You will be progressively introduced to the fundamentals of neural networks and their application in algorithmic trading. As you advance, you will build and train your own AI solution, gradually adding new features. In addition to learning MQL5, you will gain Python and OpenCL programming skills and explore integrated matrix and vector methods, which enable the solution of complex mathematical problems with concise and efficient code.

Articles on the development of trading applications

MQL5 Articles are an excellent resource for exploring the full potential of the language, covering a wide range of practical algorithmic trading tasks. For easy navigation, all articles are categorized into sections such as Example, Expert Advisors, Machine Learning and more. Every month, dozens of new articles are published on the MQL5 Algotrading community website, written by traders for other traders. Read and discuss these articles to master modern algorithmic trading. For beginners, we have compiled a list of 16 recommended articles for a quick immersion into MQL5.

Types of MQL5 Applications

MQL5 programs are divided into five specialized types based on the trading automation tasks that they implement:

- **Expert Advisor** is an automated trading system linked to a chart. An Expert Advisor contains event handlers to manage predefined events which activate execution of appropriate trading strategy elements. For example, an event of program initialization and deinitializtion, new ticks, timer events, changes in the Depth of Market, chart and custom events.
  In addition to calculating trading signals based on the implemented rules, Expert Advisors can also automatically execute trades and send them directly to a trading server. Expert Advisors are stored in <*Terminal_Directory*>*\MQL5\Experts*.
- **Custom Indicators** is a technical indicator developed by a user in addition to standard indicators integrated into the trading platform. Custom indicators, as well as standard ones, cannot trade automatically, but only implement analytical functions. Custom indicators can utilize values of other indicators for calculations, and can be called from Expert Advisors.
  Custom indicators are stored in <*Terminal_Directory*>*\MQL5\Indicators*.
- **Script** is a program for a single execution of an action. Unlike Expert Advisors, scripts do not handle any event except for trigger. A script code must contain the OnStart handler function.
  Scripts are stored in <*Terminal_DIrectory*>*\MQL5\Scripts*.
- **Service** is a program that, unlike indicators, Expert Advisors and scripts, does not require to be bound to a chart to work. Like scripts, services do not handle any event except for trigger. To launch a service, its code should contain the OnStart handler function. Services do not accept any other events except Start, but they are able to send custom events to charts using EventChartCustom. Services are stored in <terminal_directory>\MQL5\Services.
- **Library** is a set of custom functions. Libraries are intended to store and distribute commonly used algorithms of custom programs.
  Libraries are stored in <*Terminal_Directory*>*\MQL5\Libraries*.

- **Include File** is a source text of the most frequently used blocks of custom programs. Such files can be included into the source texts of Expert Advisors, scripts, custom indicators, and libraries at the compiling stage. The use of included files is more preferable than the use of libraries because of additional burden occurring at calling library functions.
  Include files can be stored in the same directory where the original file is located. In this case the #include directive with double quotes is used. Another option is to store include files in <*Terminal_Directory*>*MQL5**Include*. In this case #include with angle brackets should be used.

Language Basics

The MetaQuotes Language 5 (MQL5) is an object-oriented high-level programming language intended for writing automated trading strategies, custom technical indicators for the analysis of various financial markets. It allows not only to write a variety of expert systems, designed to operate in real time, but also create their own graphical tools to help you make trade decisions.

MQL5 is based on the concept of the popular programming language C++. As compared to MQL4, the new language now has enumerations, structures, classes and event handling. By increasing the number of embedded main types, the interaction of executable programs in MQL5 with other applications through dll is now as easy as possible. MQL5 syntax is similar to the syntax of C++, and this makes it easy to translate into it programs from modern programming languages.

To help you study the MQL5 language, all topics are grouped into the following sections:

- Syntax
- Data Types
- Operations and Expressions
- Operators
- Functions
- Variables
- Preprocessor
- Object-Oriented Programming
- Namespaces

Syntax

As to the syntax, THE MQL5 language for programming trading strategies is very much similar to the C++ programming language, except for some features:

- no address arithmetic;
- no goto operator;
- an anonymous enumeration can't be declared;
- no multiple inheritance.

See also

Enumerations, Structures and Classes, Inheritance

Comments

Multi-line comments start with the /* pair of symbols and end with the */ one. Such kind of comments cannot be nested. Single-line comments begin with the // pair of symbols and end with the newline character, they can be nested in other multi-line comments. Comments are allowed everywhere where the spaces are allowed, they can have any number of spaces in them.

Examples:

```
//--- Single-line comment
/*  Multi-
    line         // Nested single-line comment
    comment
*/
```

Identifiers

Identifiers are used as names of variables and functions. The length of the identifier can not exceed 63 characters.

Characters allowed to be written in an identifier: figures 0-9, the Latin uppercase and lowercase letters a-z and A-Z, recognized as different characters, the underscore character (_).The first character can not be a digit.

The identifier must not coincide with reserved word.

Examples:

```
NAME1 namel Total_5 Paper
```

See also

Variables, Functions

Reserved Words

The following identifiers are recorded as reserved words, each of them corresponds to a certain action, and cannot be used in another meaning:

Data Types

bool	float	uint
char	int	ulong
class	long	union
color	short	ushort
datetime	string	void
double	struct	
enum	uchar	

Access Specificators

const	private	virtual
delete	protected	
override	public	

Memory Classes

extern	input	static

Operators

break	dynamic_cast	operator
case	else	pack
continue	for	return
default	if	sizeof
delete	new	switch
do	offsetof	while

Other

this	#define	#import
true	#ifdef	#include

this	#define	#import
false	#ifndef	#property
template	#else	group
typename	#endif	namespace

Data Types

Any program operates with data. Data can be of different types depending on their purposes. For example, integer data are used to access to array components. Price data belong to those of double precision with floating point. This is related to the fact that no special data type for price data is provided in MQL5.

Data of different types are processed with different rates. Integer data are processed at the fastest. To process the double precision data, a special co-processor is used. However, because of complexity of internal representation of data with floating point, they are processed slower than the integer ones.

String data are processed at the longest because of dynamic computer memory allocation/reallocation.

The basic data types are:

- integers (char, short, int, long, uchar, ushort, uint, ulong);
- logical (bool);
- literals (ushort);
- strings (string);
- floating-point numbers (double, float);
- color (color);
- date and time (datetime);
- enumerations (enum).

Complex data types are:

- structures;
- classes.

In terms of OOP complex data types are called abstract data types.

The *color* and *datetime* types make sense only to facilitate visualization and input of parameters defined from outside - from the table of Expert Advisor or custom indicator properties (the Inputs tab). Data of *color* and *datetime* types are represented as integers. Integer types and floating-point types are called arithmetic (numeric) types.

Only implicit type casting is used in expressions, unless the explicit casting is specified.

See also

Typecasting

Integer Types

In MQL5 integers are represented by eleven types. Some types can be used together with other ones, if required by the program logic, but in this case it's necessary to remember the rules of typecasting.

The table below lists the characteristics of each type. Besides, the last column features a type in C++ corresponding to each type.

Type	Size in Bytes	Minimum Value	Maximum Value	C++ Analog
char	1	-128	127	char
uchar	1	0	255	unsigned char, BYTE
bool	1	0(false)	1(true)	bool
short	2	-32 768	32 767	short, wchar_t
ushort	2	0	65 535	unsigned short, WORD
int	4	- 2 147 483 648	2 147 483 647	int
uint	4	0	4 294 967 295	unsigned int, DWORD
color	4	-1	16 777 215	int, COLORREF
long	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	__int64
ulong	8	0	18 446 744 073 709 551 615	unsigned __int64
datetime	8	0 (1970.01.01 0:00:00)	32 535 244 799 (3000.12.31 23:59:59)	__time64_t

Integer type values can also be presented as numeric constants, color literals, date-time literals, character constants and enumerations.

Additional integer type aliases (for C/C++ compatibility)

To simplify the porting of code from other languages, such as C and C++, aliases for standard integer types were added to the MQL5 language. These aliases do not introduce new types, but are alternative names for existing types in MQL5. They can be used in all contexts where base types are applicable.

Type	Corresponds to MQL5 type	Size in Bytes	Sign	Minimum Value	Maximum Value
int8_t	char	8	signed	-128	127

Type	Corresponds to MQL5 type	Size in Bytes	Sign	Minimum Value	Maximum Value
uint8_t	uchar	8	unsigned	0	255
int16_t	short	16	signed	-32 768	32 767
uint16_t	ushort	16	unsigned	0	65 535
int32_t	int	32	signed	- 2 147 483 648	2 147 483 647
uint32_t	uint	32	unsigned	0	4 294 967 295
int64_t	long	64	signed	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
uint64_t	ulong	64	unsigned	0	18 446 744 073 709 551 615

These aliases are useful when porting existing libraries and algorithms from C/C++ to MQL5, especially in projects with a high level of platform compatibility. All new names are available in scripts, EAs and indicators.

See also

Conversion Functions, Numerical Type Constants

Char, Short, Int and Long Types

char

The *char* type takes 1 byte of memory (8 bits) and allows expressing in the binary notation 2^8=256 values. The *char* type can contain both positive and negative values. The range of values is from -128 to 127.

uchar

The *uchar* integer type also occupies 1 byte of memory, as well as the *char* type, but unlike it *uchar* is intended only for positive values. The minimum value is zero, the maximum value is 255. The first letter u in the name of the *uchar* type is the abbreviation for *unsigned*.

short

The size of the *short* type is 2 bytes (16 bits) and, accordingly, it allows expressing the range of values equal to 2 to the power 16: 2^16 = 65 536.Since the *short* type is a signed one, and contains both positive and negative values, the range of values is between -32 768 and 32 767.

ushort

The unsigned *short* type is the type *ushort*, which also has a size of 2 bytes. The minimum value is 0, the maximum value is 65 535.

int

The size of the *int* type is 4 bytes (32 bits). The minimal value is -2 147 483 648, the maximal one is 2 147 483 647.

uint

The unsigned integer type is *uint*. It takes 4 bytes of memory and allows expressing integers from 0 to 4 294 967 295.

long

The size of the *long* type is 8 bytes (64 bits). The minimum value is -9 223 372 036 854 775 808, the maximum value is 9 223 372 036 854 775 807.

ulong

The *ulong* type also occupies 8 bytes and can store values from 0 to 18 446 744 073 709 551 615.

Examples:

```
char  ch=12;
short sh=-5000;
int   in=2445777;
```

Since the unsigned integer types are not designed for storing negative values, the attempt to set a negative value can lead to unexpected consequences. Such a simple script will lead to an infinite loop:

```
//--- Infinite loop
void OnStart()
  {
   uchar  u_ch;

   for(char ch=-128;ch<128;ch++)
     {
      u_ch=ch;
      Print("ch = ",ch," u_ch = ",u_ch);
     }
  }
```

The correct variant is:

```
//--- Correct variant
void OnStart()
  {
   uchar  u_ch;

   for(char ch=-128;ch<=127;ch++)
     {
      u_ch=ch;
      Print("ch = ",ch," u_ch = ",u_ch);
      if(ch==127) break;
     }
  }
```

Result:

```
   ch= -128  u_ch= 128
   ch= -127  u_ch= 129
   ch= -126  u_ch= 130
   ch= -125  u_ch= 131
   ch= -124  u_ch= 132
   ch= -123  u_ch= 133
   ch= -122  u_ch= 134
   ch= -121  u_ch= 135
   ch= -120  u_ch= 136
   ch= -119  u_ch= 137
   ch= -118  u_ch= 138
   ch= -117  u_ch= 139
   ch= -116  u_ch= 140
   ch= -115  u_ch= 141
   ch= -114  u_ch= 142
   ch= -113  u_ch= 143
   ch= -112  u_ch= 144
   ch= -111  u_ch= 145
   ...
```

Examples:

```
//--- Negative values can not be stored in unsigned types
uchar  u_ch=-120;
ushort u_sh=-5000;
uint   u_in=-401280;
```

Hexadecimal: numbers 0-9, the letters a-f or A-F for the values of 10-15; start with 0x or 0X.

Examples:

```
0x0A, 0x12, 0X12, 0x2f, 0xA3, 0Xa3, 0X7C7
```

For integer variables, the values can be set in binary form using B prefix. For example, you can encode the working hours of a trading session into int type variable and use information about them according to the required algorithm:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set 1 for working hours and 0 for nonworking ones
   int AsianSession   =B'111111111'; // Asian session from 0:00 to 9:00
   int EuropeanSession=B'111111111000000000'; // European session 9:00 - 18:00
   int AmericanSession =B'111111110000000000000011'; // American session 16:00 - 02:0
//--- derive numerical values of the sessions
   PrintFormat("Asian session hours as value =%d",AsianSession);
   PrintFormat("European session hours as value is %d",EuropeanSession);
   PrintFormat("American session hours as value is %d",AmericanSession);
//--- and now let's display string representations of the sessions' working hours
   Print("Asian session ",GetHoursForSession(AsianSession));
   Print("European session ",GetHoursForSession(EuropeanSession));
   Print("American session ",GetHoursForSession(AmericanSession));
//---
  }
//+------------------------------------------------------------------+
//| return the session's working hours as a string                   |
//+------------------------------------------------------------------+
string GetHoursForSession(int session)
  {
//--- in order to check, use AND bit operations and left shift by 1 bit <<=1
//--- start checking from the lowest bit
   int bit=1;
   string out="working hours: ";
//--- check all 24 bits starting from the zero and up to 23 inclusively
   for(int i=0;i<24;i++)
     {
      //--- receive bit state in number
      bool workinghour=(session&bit)==bit;
      //--- add the hour's number to the message
      if(workinghour )out=out+StringFormat("%d ",i);
      //--- shift by one bit to the left to check the value of the next one
      bit<<=1;
     }
//--- result string
   return out;
  }
```

See also

Typecasting

Character Constants

Characters as elements of a string in MQL5 are indexes in the Unicode character set. They are hexadecimal values that can be cast into integers, and that can be manipulated by integer operations like addition and subtraction.

Any single character in quotation marks or a hexadecimal ASCII code of a character as '\x10' is a character constant and is of ushort type. For example, a record of '0' type is a numerical value 30, that corresponds to the index of zero in the table of characters.

Example:

```
void OnStart()
  {
//--- define character constants
   int symbol_0='0';
   int symbol_9=symbol_0+9; // get symbol '9'
//--- output values of constants
   printf("In a decimal form: symbol_0 = %d,  symbol_9 = %d",symbol_0,symbol_9);
   printf("In a hexadecimal form: symbol_0 = 0x%x,  symbol_9 = 0x%x",symbol_0,symbol_
//--- enter constants into a string
   string test="";
   StringSetCharacter(test,0,symbol_0);
   StringSetCharacter(test,1,symbol_9);
//--- this is what they look like in a string
   Print(test);
  }
```

A backslash is a control character for a compiler when dealing with constant strings and character constants in a source text of a program. Some symbols, for example a single quote ('), double quotes ("), backslash (\) and control characters can be represented as a combination of symbols that start with a backslash (\), according to the below table:

Character name	Mnemonic code or image	Record in MQL5	Numeric value
new line (line feed)	LF	'\n'	10
horizontal tab	HT	'\t'	9
carriage return	CR	'\r'	13
backslash	\	'\\'	92
single quote	'	'\''	39
double quote	"	'\"'	34
hexadecimal code	hhhh	'\xhhhh'	1 to 4 hexadecimal characters
decimal code	d	'\d'	decimal number from 0 to 65535

If a backslash is followed by a character other than those described above, result is undefined.

Example

```
void OnStart()
  {
//--- declare character constants
   int a='A';
   int b='$';
   int c='©';      // code 0xA9
   int d='\xAE';   // code of the symbol ®
//--- output print constants
   Print(a,b,c,d);
//--- add a character to the string
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,b);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,c);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,d);
   Print(test);
//--- represent characters as a number
   int a1=65;
   int b1=36;
   int c1=169;
   int d1=174;
//--- add a character to the string
   StringSetCharacter(test,1,a1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,b1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,c1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,d1);
   Print(test);
  }
```

As it was mentioned above, the value of a character constant (or variable) is an index in the table of characters. Index being an integer, it can be written in different ways.

```
void OnStart()
  {
```

```
//---
   int a=0xAE;     // the code of ® corresponds to the '\xAE' literal
   int b=0x24;     // the code of $ corresponds to the '\x24' literal
   int c=0xA9;     // the code of © corresponds to the '\xA9' literal
   int d=0x263A;   // the code of ☺ corresponds to the '\x263A' literal
//--- show values
   Print(a,b,c,d);
//--- add a character to the string
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,b);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,c);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,d);
   Print(test);
//--- codes of suits
   int a1=0x2660;
   int b1=0x2661;
   int c1=0x2662;
   int d1=0x2663;
//--- add a character of spades
   StringSetCharacter(test,1,a1);
   Print(test);
//--- add a character of hearts
   StringSetCharacter(test,2,b1);
   Print(test);
//--- add a character of diamonds
   StringSetCharacter(test,3,c1);
   Print(test);
//--- add a character of clubs
   StringSetCharacter(test,4,d1);
   Print(test);
//--- Example of character literals in a string
   test="Queen\x2660Ace\x2662";
   printf("%s",test);
  }
```

The internal representation of a character literal is the ushort type. Character constants can accept values from 0 to 65535.

See also

StringSetCharacter(), StringGetCharacter(), ShortToString(), ShortArrayToString(), StringToShortArray()

Datetime Type

The datetime type is intended for storing the date and time as the number of seconds elapsed since January 01, 1970. This type occupies 8 bytes of memory.

Constants of the date and time can be represented as a literal string, which consists of 6 parts showing the numerical value of the year, month, day (or day, month, year), hours, minutes and seconds. The constant is enclosed in single quotation marks and starts with the D character.

Values range from 1 January, 1970 to 31 December, 3000. Either date (year , month, day) or time (hours, minutes, seconds), or all together can be omitted.

With literal date specification, it is desirable that you specify year, month and day. Otherwise the compiler returns a warning about an incomplete entry.

Examples:

```
datetime NY=D'2015.01.01 00:00';     // Time of beginning of year 2015
datetime d1=D'1980.07.19 12:30:27';  // Year Month Day Hours Minutes Seconds
datetime d2=D'19.07.1980 12:30:27';  // Equal to D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';        // Equal to D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';           // Equal to D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;             // Compilation date
datetime compilation_date_time=__DATETIME__;    // Compilation date and time
datetime compilation_time=__DATETIME__-__DATE__;// Compilation time
//--- Examples of declarations after which compiler warnings will be returned
datetime warning1=D'12:30:27';       // Equal to D'[date of compilation] 12:30:27'
datetime warning2=D'';               // Equal to __DATETIME__
```

See also

Structure of the Date Type, Date and Time, TimeToString, StringToTime

Color Type

The color type is intended for storing information about color and occupies 4 bytes in memory. The first byte is ignored, the remaining 3 bytes contain the RGB-components.

Color constants can be represented in three ways: literally, by integers, or by name (for named Web-colors only).

Literal representation consists of three parts representing numerical rate values of the three main color components: red, green, blue. The constant starts with C and is enclosed in single quotes. Numerical rate values of a color component lie in the range from 0 to 255.

Integer-valued representation is written in a form of hexadecimal or a decimal number. A hexadecimal number looks like 0x00BBGGRR, where RR is the rate of the red color component, GG - of the green one, and BB - of the blue one. Decimal constants are not directly reflected in the RGB. They represent a decimal value of the hexadecimal integer representation.

Specific colors reflect the so-called Web-colors set.

Examples:

```
//--- Literals
C'128,128,128'    // Gray
C'0x00,0x00,0xFF' // Blue
//color names
clrRed               // Red
clrYellow            // Yellow
clrBlack             // Black
//--- Integral representations
0xFFFFFF          // White
16777215          // White
0x008000          // Green
32768             // Green
```

See also

Web Colors, ColorToString, StringToColor, Typecasting

Bool Type

The bool type is intended to store the logical values of true or false, numeric representation of them is 1 or 0, respectively.

Examples:

```
bool a = true;
bool b = false;
bool c = 1;
```

The internal representation is a whole number 1 byte large. It should be noted that in logical expressions you can use other integer or real types or expressions of these types - the compiler will not generate any error. In this case, the zero value will be interpreted as false, and all other values - as true.

Examples:

```
   int i=5;
   double d=-2.5;
   if(i) Print("i = ",i," and is set to true");
   else Print("i = ",i," and is set to false");

   if(d) Print("d = ",d," and has the true value");
   else Print("d = ",d," and has the false value");

   i=0;
   if(i) Print("i = ",i," and has the true value");
   else Print("i = ",i," and has the false value");

   d=0.0;
   if(d) Print("d = ",d," and has the true value");
   else Print("d = ",d," and has the false value");

//--- Execution results
//   i= 5 and has the true value
//   d= -2.5 and has the true value
//   i= 0 and has the false value
//   d= 0 and has the false value
```

See also

Boolean Operations, Precedence Rules

Enumerations

Data of the enum type belong to a certain limited set of data. Defining the enumeration type:

```
enum name of enumerable type
  {
   list of values
  };
```

The list of values is a list of identifiers of named constants separated by commas.

Example:

```
enum months  // enumeration of named constants
   {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
   };
```

After the enumeration is declared, a new integer-valued 4-byte data type appears. Declaration of the new data type allows the compiler to strictly control types of passed parameters, because enumeration introduces new named constants. In the above example, the January named constant has the value of 0, February - 1, December - 11.

Rule: If a certain value is not assigned to a named constant that is a member of the enumeration, its new value will be formed automatically. If it is the first member of the enumeration, the 0 value will be assigned to it. For all subsequent members, values will be calculated based on the value of the previous members by adding one.

Example:

```
enum intervals  // Enumeration of named constants
   {
    month=1,     // Interval of one month
    two_months,  // Two months
    quarter,     // Three months - quarter
    halfyear=6,  // Half a year
    year=12,     // Year - 12 months
   };
```

Notes

- Unlike C++, the size of the internal representation of the enumerated type in MQL5 is always equal to 4 bytes. That is, sizeof (months) returns the value 4.
- Unlike C++, an anonymous enumeration can't be declared in MQL5. That is, a unique name must be always specified after the enum keyword.

See also

Typecasting

Real Types (double, float)

Real types (or floating-point types) represent values with a fractional part. In the MQL5 language there are two types for floating point numbers.The method of representation of real numbers in the computer memory is defined by the IEEE 754 standard and is independent of platforms, operating systems or programming languages.

Type	Size in bytes	Minimal Positive Value	Maximum Value	C++ Analog
float	4	1.175494351e-38	3.402823466e+38	float
double	8	2.2250738585072014e-308	1.7976931348623158e+308	double

double

double real number type occupies 64 bits (1 sign bit, 11 exponent bits and 52 mantissa bits).

float

float real number type occupies 32 bits (1 sign bit, 8 exponent bits and 23 mantissa bits).

vector

One-dimensional array of double type numbers. Memory for data is allocated dynamically. Vector properties can be obtained using the methods, while the vector size can be changed. The vector<double> entry can be used in template functions.

vectorf

One-dimensional array of float type numbers can be used instead of vector if the loss of precision does not matter. The vector<float> entry can be used in template functions.

vectorc

One-dimensional array of complex type numbers is meant to handle complex numbers. The vector<complex> entry can be used in template functions. Operations on vectorc type vectors are not implemented yet.

matrix

Matrix is a two-dimensional array of double type numbers. Memory for matrix elements is distributed dynamically. Matrix properties can be obtained using the methods, while the matrix shape can be changed. The matrix<double> entry can be used in template functions.

matrixf

Two-dimensional array of float type numbers can be used instead of matrix if the loss of precision does not matter. The matrix<float> entry can be used in template functions.

matrixc

Two-dimensional array of complex type numbers is meant to handle complex numbers. The matrix<complex> entry can be used in template functions. Operations on matrixc type matrices are not implemented yet.

The double name means that the accuracy of these numbers is twice the accuracy of the float type numbers. In most cases, the double type is the most convenient one. In many cases the limited precision of float numbers is not enough. The reason why the float type is still used is saving the memory (this is important for large arrays of real numbers).

Floating-point constants consist of an integer part, a point (.) and the fractional part. The integer and fractional parts are sequences of decimal digits.

Examples:

```
   double a=12.111;
   double b=-956.1007;
   float  c =0.0001;
   float  d =16;
```

There is a scientific way of writing real constants, often this method of recording is more compact than the traditional one.

Example:

```
   double c1=1.12123515e-25;
   double c2=0.000000000000000000000000112123515; // 24 zero after the decimal point
 
   Print("1. c1 =",DoubleToString(c1,16));
   // Result: 1. c1 = 0.0000000000000000
 
   Print("2. c1 =",DoubleToString(c1,-16));
   // Result: 2. c1 = 1.1212351499999999e-025
 
   Print("3. c2 =",DoubleToString(c2,-16));
   // Result: 3. c2 = 1.1212351499999999e-025
```

It should be remembered that real numbers are stored in memory with some limited accuracy in the binary system, while generally the decimal notation is used. That's why many numbers that are precisely represented in the decimal system can be written only as an infinite fraction in the binary system.

For example, numbers 0.3 and 0.7 are represented in the computer as infinite fractions, while the number of 0.25 is stored exactly, because it represents the power of two.

In this regard, it is strongly recommended not to compare two real numbers for equality, because such a comparison is not correct.

Example:

```
void OnStart()
  {
//---
   double three=3.0;
   double x,y,z;
   x=1/three;
   y=4/three;
   z=5/three;
   if(x+y==z)
      Print("1/3 + 4/3 == 5/3");
   else
      Print("1/3 + 4/3 != 5/3");
// Result: 1/3 + 4/3 != 5/3
  }
```

If you still need to compare the equality of two real numbers, then you can do this in two different ways. The first way is to compare the difference between two numbers with some small quantity that specifies the accuracy of comparison.

Example:

```
bool EqualDoubles(double d1,double d2,double epsilon)
  {
   if(epsilon<0)
      epsilon=-epsilon;
//---
   if(d1-d2>epsilon)
      return false;
   if(d1-d2<-epsilon)
      return false;
//---
   return true;
  }
void OnStart()
  {
   double d_val=0.7;
   float  f_val=0.7;
   if(EqualDoubles(d_val,f_val,0.000000000000001))
      Print(d_val," equals ",f_val);
   else
      Print("Different: d_val = ",DoubleToString(d_val,16),"  f_val = ",DoubleToStrin
// Result: Different: d_val= 0.7000000000000000   f_val= 0.6999999880790710
  }
```

Note that the value of epsilon in the above example can not be less than the predefined constant DBL_EPSILON. The value of this constant is 2.2204460492503131e-016. The constant corresponding to the float type is FLT_EPSILON = 1.192092896e-07. The meaning of these values is the following: it is the lowest value that satisfies the condition 1.0 + DBL_EPSILON! = 1.0 (for numbers of float type 1.0 + FLT_EPSILON! = 1.0).

The second way offers comparing the normalized difference of two real numbers with zero. It's meaningless to compare the difference of normalized numbers with a zero, because any mathematical operation with normalized numbers gives a non-normalized result.

Example:

```
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0)
      return(true);
   else
      return(false);
  }
void OnStart()
  {
   double d_val=0.3;
   float  f_val=0.3;
   if(CompareDoubles(d_val,f_val))
      Print(d_val," equals ",f_val);
   else
      Print("Different: d_val = ",DoubleToString(d_val,16),"  f_val = ",DoubleToStrin
// Result: Different: d_val= 0.3000000000000000   f_val= 0.3000000119209290
  }
```

Some operations of the mathematical co-processor can result in the invalid real number, which can't be used in mathematical operations and operations of comparison, because the result of operations with invalid real numbers is undefined. For example, when trying to calculate the arcsine of 2, the result is the negative infinity.

Example:

```
   double abnormal = MathArcsin(2.0);
   Print("MathArcsin(2.0) =",abnormal);
// Result:  MathArcsin(2.0) = -1.#IND
```

Besides the minus infinity there is the plus infinity and NaN (not a number). To determine that this number is invalid, you can use MathIsValidNumber(). According to the IEEE standard, they have a special machine representation. For example, plus infinity for the double type has the bit representation of 0x7FF0 0000 0000 0000.

Examples:

```
struct str1
  {
   double d;
  };
struct str2
  {
   long l;
  };

//--- Start
   str1 s1;
   str2 s2;
//---
   s1.d=MathArcsin(2.0);        // Get the invalid number -1.#IND
   s2=s1;
   printf("1.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFFF000000000000;     // invalid number -1.#QNAN
   s1=s2;
   printf("2.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF7000000000000;     // greatest non-number SNaN
   s1=s2;
   printf("3.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF8000000000000;     // smallest non-number QNaN
   s1=s2;
   printf("4.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FFF000000000000;     // greatest non-number QNaN
   s1=s2;
   printf("5.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF0000000000000;     // Positive infinity 1.#INF and smallest non-number S
   s1=s2;
   printf("6.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFF0000000000000;     // Negative infinity -1.#INF
   s1=s2;
   printf("7.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x8000000000000000;     // Negative zero -0.0
   s1=s2;
   printf("8.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FE0000000000000;     // 0.5
   s1=s2;
   printf("9.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FF0000000000000;     // 1.0
   s1=s2;
   printf("10.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FEFFFFFFFFFFFFF;     // Greatest normalized number (MAX_DBL)
   s1=s2;
   printf("11.  %.16e %I64X",s1.d,s2.l);
//---
   s2.l=0x0010000000000000;     // Smallest positive normalized (MIN_DBL)
   s1=s2;
   printf("12.  %.16e %.16I64X",s1.d,s2.l);
//---
   s1.d=0.7;                    // Show that the number of 0.7 - endless fraction
   s2=s1;
   printf("13.  %.16e %.16I64X",s1.d,s2.l);
/*
1.  -1.#IND00 FFF8000000000000
2.  -1.#QNAN0 FFFF000000000000
3.   1.#SNAN0 7FF7000000000000
4.   1.#QNAN0 7FF8000000000000
5.   1.#QNAN0 7FFF000000000000
6.   1.#INF00 7FF0000000000000
7.  -1.#INF00 FFF0000000000000
8.  -0.000000 8000000000000000
9.   0.500000 3FE0000000000000
10.  1.000000 3FF0000000000000
11.  1.7976931348623157e+308 7FEFFFFFFFFFFFFF
12.  2.2250738585072014e-308 0010000000000000
13.  6.9999999999999996e-001 3FE6666666666666
*/
```

See also

DoubleToString, NormalizeDouble, Numeric Type Constants

Complex number (complex)

The built-in complex type is a structure with two double fields:

```
struct complex
  {
   double             real;   // Real part
   double             imag;   // Imaginary part
  };
```

The "complex" type can be passed by value as a parameter for MQL5 functions (in contrast to ordinary structures, which are only passed by reference). For functions imported from DLLs, the "complex" type must be passed only by reference.

The 'i' suffix is used to describe complex constants:

```
complex square(complex c)
  {
   return(c*c);
  }
void OnStart()
  {
   Print(square(1+2i));  // A constant is passed as a parameter
  }
// "(-3,4)" will be output, which is a string representation of the complex number
```

Only simple operations are currently available for complex numbers: =, +, -, *, /, +=, -=, *=, /=, ==,!=.

Support for additional mathematical functions will be added later, enabling the calculation of the absolute value, sine, cosine and others.

vectorc

One-dimensional array of complex type numbers is meant to handle complex numbers. The vector<complex> entry can be used in template functions. Operations on vectorc type vectors are not implemented yet.

matrixc

Two-dimensional array of complex type numbers is meant to handle complex numbers. The matrix<complex> entry can be used in template functions. Operations on matrixc type matrices are not implemented yet.

String Type

The string type is used for storing text strings. A text string is a sequence of characters in the Unicode format with the final zero at the end of it. A string constant can be assigned to a string variable. A string constant is a sequence of Unicode characters enclosed in double quotes: "This is a string constant".

If you need to include a double quote (") into a string, the backslash character (\) must be put before it. Any special character constants can be written in a string, if the backslash character (\) is typed before them.

Examples:

```
string svar="This is a character string";
string svar2=StringSubstr(svar,0,4);
Print("Copyright symbol\t\x00A9");
FileWrite(handle,"This string contains a new line symbols \n");
string MT5path="C:\\Program Files\\MetaTrader 5";
```

To make the source code readable, long constant strings can be split into parts without addition operation. During compilation, these parts will be combined into one long string:

```
//--- Declare a long constant string
   string HTML_head="<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\""
                    " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n"
                    "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
                    "<head>\n"
                    "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ut
                    "<title>Trade Operations Report</title>\n"
                    "</head>";
//--- Output the constant string into log
   Print(HTML_head);
  }
```

Built-in string type methods

Strings can be handled by string functions, conversion functions and built-in methods of string type provided in the table:

String method	Analog	Description
Constructor string(const int len)		Constructs a string of a specified
string[] length both for reading and writing. The index should be within BufferSize()		Provides access to the string element by a specified index

String method	Analog	Description
static string string.Init(const int len, const ushort character);	StringInit	Initializes a string using specified symbols with a specified size
void string.Fill(const ushort character);	StringFill	Fills in the string with a specified symbol
int string.Len();	StringLen	Returns the number of characters in a string
int string.BufferSize();	StringBufferLen	Returns the size of a buffer distributed for a string
bool string.SetLen(const int new_len);	StringSetLength	Sets a specified length (in characters) for a string
bool string.Reserve(const int buffer_len);	StringReserve	Reserves the buffer of a specified size for a string in memory
bool string.Add(const string substring);	StringAdd	Adds a specified substring to the end
int string.Concatenate(const scalar val1, const scalar val2...);	StringConcatenate	Forms a string consisting of passed parameters
array string.Split(const ushort separator, const bool long_separator);	StringSplit	Returns a string array by a specified separator
int string.Compare(const string str, const bool case_sensivity);	StringCompare	Compares with a specified string and returns 1 if the first string exceeds the second one; 0 - if the strings are equal; -1 (minus one) - if the first string is less than the second one
string string.Substr(const int start_pos, const int len);	StringSubstr	Retrieves a substring from a specified position
int string.Find(const string substr, const int pos);	StringFind	Returns an index of the position the necessary substring starts from
void string.ToLower();	StringToLower	Converts all characters to lower case
void string.ToUpper();	StringToUpper	Converts all characters to upper case
int string.TrimLeft();	StringTrimLeft	Deletes spaces, as well as carriage movement and tabulation characters to the left
int string.TrimRight()	StringTrimRight	Deletes spaces, as well as carriage movement and tabulation characters to the right
void string.Double(const double var, const int digits=8);	DoubleToString	Converts a string to a double type number

String method	Analog	Description
void string.Enum(const enum value);	EnumToString	Converts the enumeration value of any type into a string
void string.Integer(const int value, const int str_len=0, const ushort fill=' ');	IntegerToString	Converts a string to a long type number
void string.CharArray(const uchar array[], const int start_pos=0, const int len=-1, const uint cp=CP_ACP);	CharArrayToString	Converts part of the uchar type array to a string
void string.ShortArray(const ushort array[], const int start_pos=0, const int len=-1);	ShortArrayToString	Copies part of the ushort type array to a string
void string.Time(const datetime dt,const int mode=TIME_DATE|TIME_MINUTES);	TimeToString	Converts datetime to the "yyyy.mm.dd hh:mi" format string.
void string.Format(const string format_str);	StringFormat	Formats the obtained parameters into a string

See also

Conversion Functions, String Functions, FileOpen, FileReadString, FileWriteString

Structures, Classes and Interfaces

Structures

A structure is a set of elements of any type (except for the void type). Thus, the structure combines logically related data of different types.

Structure Declaration

The structure data type is determined by the following description:

```
struct structure_name
  {
   elements_description
  };
```

The structure name can't be used as an identifier (name of a variable or function). It should be noted that in MQL5 structure elements follow one another directly, without alignment. In C++ such an order is made to the compiler using the following instruction:

```
#pragma pack(1)
```

If you want to do another alignment in the structure, use auxiliary members, "fillers" to the right size.

Example:

```
struct trade_settings
  {
   uchar  slippage;     // value of the permissible slippage-size 1 byte
   char   reserved1;    // skip 1 byte
   short  reserved2;    // skip 2 bytes
   int    reserved4;    // another 4 bytes are skipped. ensure alignment of the bounda
   double take;         // values of the price of profit fixing
   double stop;         // price value of the protective stop
  };
```

Such a description of aligned structures is necessary only for transferring to imported dll-functions.

Attention: This example illustrates incorrectly designed data. It would be better first to declare the *take* and *stop* large data of the double type, and then declare the *slippage* member of the uchar type. In this case, the internal representation of data will always be the same regardless of the value specified in *#pragma pack()*.

If a structure contains variables of the string type and/or object of a dynamic array, the compiler assigns an implicit constructor to such a structure. This constructor resets all the structure members of string type and correctly initializes objects of the dynamic array.

Simple Structures

Structures that do not contain strings, class objects, pointers and objects of dynamic arrays are called simple structures. Variables of simple structures, as well as their arrays can be passed as parameters to functions imported from DLL.

Copying of simple structures is allowed only in two cases:

- If the objects belong to the same structure type
- if the objects are connected by the lineage meaning that one structure is a descendant of another.

To provide an example, let's develop the CustomMqlTick custom structure with its contents identical to the built-in MqlTick one. The compiler does not allow copying the MqlTick object value to the CustomMqlTick type object. Direct typecasting to the necessary type also causes the compilation error:

```
      //--- copying simple structures of different types is forbidden
      my_tick1=last_tick;               // compiler returns an error here
 
      //--- typecasting structures of different types to each other is forbidden as we
      my_tick1=(CustomMqlTick)last_tick;// compiler returns an error here
```

Therefore, only one option is left – copying the values of the structure elements one by one. It is still allowed to copy the values of the same type of CustomMqlTick.

```
      CustomMqlTick my_tick1,my_tick2;
      //--- it is allowed to copy the objects of the same type of CustomMqlTick the fo
      my_tick2=my_tick1;
 
      //--- create an array out of the objects of the simple CustomMqlTick structure a
      CustomMqlTick arr[2];
      arr[0]=my_tick1;
      arr[1]=my_tick2;
```

The ArrayPrint() function is called for a check to display the *arr[] array value in the journal*.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- develop the structure similar to the built-in MqlTick
   struct CustomMqlTick
     {
      datetime          time;          // Last price update time
      double            bid;           // Current Bid price
      double            ask;           // Current Ask price
      double            last;          // Current price of the last trade (Last)
      ulong             volume;        // Volume for the current Last price
      long              time_msc;      // Last price update time in milliseconds
      uint              flags;         // Tick flags
     };
   //--- get the last tick value
   MqlTick last_tick;
   CustomMqlTick my_tick1,my_tick2;
//--- attempt to copy data from MqlTick to CustomMqlTick
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      //--- copying unrelated simple structures is forbidden
      //1. my_tick1=last_tick;               // compiler returns an error here

      //--- typecasting unrelated structures to each other is forbidden as well
      //2. my_tick1=(CustomMqlTick)last_tick;// compiler returns an error here

      //--- therefore, copy the structure members one by one
      my_tick1.time=last_tick.time;
      my_tick1.bid=last_tick.bid;
      my_tick1.ask=last_tick.ask;
      my_tick1.volume=last_tick.volume;
      my_tick1.time_msc=last_tick.time_msc;
      my_tick1.flags=last_tick.flags;

      //--- it is allowed to copy the objects of the same type of CustomMqlTick the fo
      my_tick2=my_tick1;

      //--- create an array out of the objects of the simple CustomMqlTick structure a
      CustomMqlTick arr[2];
      arr[0]=my_tick1;
      arr[1]=my_tick2;
      ArrayPrint(arr);
//--- example of displaying values of the array containing the objects of CustomMqlTic
      /*
                       [time]    [bid]    [ask]    [last] [volume]     [time_msc] [flags]
      [0] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000  1450000 1496070277157       2
      [1] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000  1450000 1496070277157       2
      */
     }
   else
      Print("SymbolInfoTick() failed, error = ",GetLastError());
  }
```

The second example shows the features of copying simple structures by the lineage. Suppose that we have the Animal basic structure, from which the Cat and Dog structures are derived. We can copy the Animal and Cat objects, as well as the Animal and Dog objects to each other but we cannot copy Cat and Dog to each other, although both are descendants of the Animal structure.

```
//--- structure for describing dogs
struct Dog: Animal
  {
   bool              hunting;       // hunting breed
  };
//--- structure for describing cats
struct Cat: Animal
  {
   bool              home;          // home breed
  };
//--- create objects of child structures
```

```
   Dog dog;
   Cat cat;
//--- can be copied from ancestor to descendant (Animal ==> Dog)
   dog=some_animal;
   dog.swim=true;    // dogs can swim
//--- you cannot copy objects of child structures (Dog != Cat)
   cat=dog;        // compiler returns an error
```

Complete example code:

```
//--- basic structure for describing animals
struct Animal
  {
   int               head;          // number of heads
   int               legs;          // number of legs
   int               wings;         // number of wings
   bool              tail;          // tail
   bool              fly;           // flying
   bool              swim;          // swimming
   bool              run;           // running
  };
//--- structure for describing dogs
struct Dog: Animal
  {
   bool              hunting;       // hunting breed
  };
//--- structure for describing cats
struct Cat: Animal
  {
   bool              home;          // home breed
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create and describe an object of the basic Animal type
   Animal some_animal;
   some_animal.head=1;
   some_animal.legs=4;
   some_animal.wings=0;
   some_animal.tail=true;
   some_animal.fly=false;
   some_animal.swim=false;
   some_animal.run=true;
//--- create objects of child types
   Dog dog;
   Cat cat;
//--- can be copied from ancestor to descendant (Animal ==> Dog)
```

```
   dog=some_animal;
   dog.swim=true;     // dogs can swim
//--- you cannot copy objects of child structures (Dog != Cat)
   //cat=dog;         // compiler returns an error here
//--- therefore, it is possible to copy elements one by one only
   cat.head=dog.head;
   cat.legs=dog.legs;
   cat.wings=dog.wings;
   cat.tail=dog.tail;
   cat.fly=dog.fly;
   cat.swim=false;    // cats cannot swim
//--- it is possible to copy the values from descendant to ancestor
   Animal elephant;
   elephant=cat;
   elephant.run=false;// elephants cannot run
   elephant.swim=true;// elephants can swim
//--- create an array
   Animal animals[4];
   animals[0]=some_animal;
   animals[1]=dog;
   animals[2]=cat;
   animals[3]=elephant;
//--- print out
   ArrayPrint(animals);
//--- execution result
/*
       [head] [legs] [wings] [tail] [fly] [swim] [run]
   [0]      1      4       0   true false  false  true
   [1]      1      4       0   true false   true  true
   [2]      1      4       0   true false  false false
   [3]      1      4       0   true false   true false
*/
  }
```

Another way to copy simple types is using a union. The objects of the structures should be members of the same union - see the example in union.

Access to Structure Members

The name of a structure becomes a new data type, so you can declare variables of this type. The structure can be declared only once within a project. The structure members are accessed using the point operation (.).

Example:

```
struct trade_settings
  {
   double take;         // values of the profit fixing price
   double stop;         // value of the protective stop price
   uchar  slippage;     // value of the acceptable slippage
  };
//--- create up and initialize a variable of the trade_settings type
trade_settings my_set={0.0,0.0,5};
if (input_TP>0) my_set.take=input_TP;
```

'pack' for aligning structure and class fields

The special pack attribute allows setting the alignment of structure or class fields.

```
 pack([n])
```

where n is one of the following values: 1, 2, 4, 8 or 16. It may be absent.

Example:

```
   struct pack(sizeof(long)) MyStruct
     {
      // structure members are to be aligned to the 8-byte boundary
     };
or
   struct MyStruct pack(sizeof(long))
     {
      // structure members are to be aligned to the 8-byte boundary
     };
```

'pack(1)' is applied by default for structures. This means that the structure members are located one after another in memory, and the structure size is equal to the sum of its members' size.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- simple structure with no alignment
   struct Simple_Structure
     {
      char              c; // sizeof(char)=1
      short             s; // sizeof(short)=2
      int               i; // sizeof(int)=4
      double            d; // sizeof(double)=8
     };
   //--- declare a simple structure instance
   Simple_Structure s;
//--- display the size of each structure member
   Print("sizeof(s.c)=",sizeof(s.c));
   Print("sizeof(s.s)=",sizeof(s.s));
   Print("sizeof(s.i)=",sizeof(s.i));
   Print("sizeof(s.d)=",sizeof(s.d));
//--- make sure the size of POD structure is equal to the sum of its members' size
```

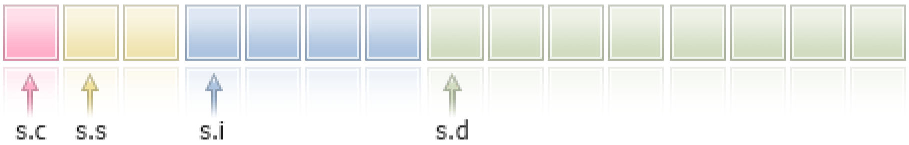
```
   Print("sizeof(simple_structure)=",sizeof(simple_structure));
/*
  Result:
   sizeof(s.c)=1
   sizeof(s.s)=2
   sizeof(s.i)=4
   sizeof(s.d)=8
   sizeof(simple_structure)=15
*/
  }
```

Alignment of the structure fields may be needed when exchanging data with third-party libraries (*.DLL) where such alignment is applied.

Let's use some examples to show how alignment works. We will apply a structure consisting of four members with no alignment.

```
//--- simple structure with no alignment
   struct Simple_Structure pack() // no size is specified, alignment to the boundary
     {
      char               c; // sizeof(char)=1
      short              s; // sizeof(short)=2
      int                i; // sizeof(int)=4
      double             d; // sizeof(double)=8
     };
//--- declare a simple structure instance
   Simple_Structure s;
```

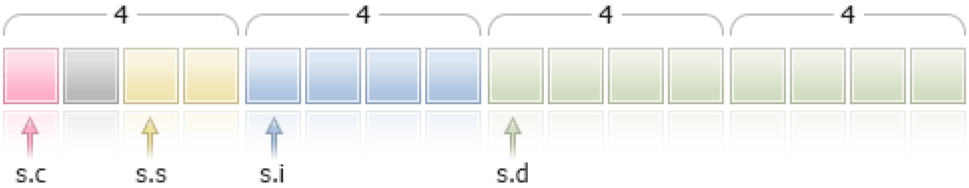
Structure fields are to be located in memory one after another according to the declaration order and type size. The structure size is 15, while an offset to the structure fields in the arrays is undefined.



Now declare the same structure with the alignment of 4 bytes and run the code.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- simple structure with the 4-byte alignment
   struct Simple_Structure pack(4)
     {
      char               c; // sizeof(char)=1
      short              s; // sizeof(short)=2
      int                i; // sizeof(int)=4
      double             d; // sizeof(double)=8
     };
   //--- declare a simple structure instance
   Simple_Structure s;
//--- display the size of each structure member
   Print("sizeof(s.c)=",sizeof(s.c));
   Print("sizeof(s.s)=",sizeof(s.s));
   Print("sizeof(s.i)=",sizeof(s.i));
   Print("sizeof(s.d)=",sizeof(s.d));
//--- make sure the size of POD structure is now not equal to the sum of its members'
   Print("sizeof(simple_structure)=",sizeof(simple_structure));
/*
  Result:
   sizeof(s.c)=1
   sizeof(s.s)=2
   sizeof(s.i)=4
   sizeof(s.d)=8
   sizeof(simple_structure)=16 // structure size has changed
*/
  }
```

The structure size has changed so that all members of 4 bytes and more has an offset from the beginning of the structure multiple of 4 bytes. Smaller members are to be aligned to their own size boundary (for example, 2 for 'short'). This is how it looks (the added byte is shown in gray).



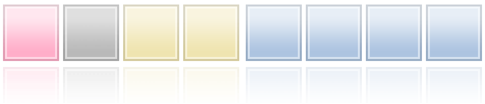
In this case, 1 byte is added after the **s.c** member, so that the **s.s** (sizeof(short)==2) field has the boundary of 2 bytes (alignment for 'short' type).

The offset to the beginning of the structure in the array is also aligned to the 4-byte boundary, i.e. the addresses of the a[0], a[1] and a[n] elements are to be multiple of 4 bytes for Simple_Structure arr[].

Let's consider two more structures consisting of similar types with 4-bytes alignment but different member order. In the first structure, the members are located in type size ascending order.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- simple structure aligned to the 4-byte boundary
   struct CharShortInt pack(4)
     {
      char               c; // sizeof(char)=1
      short              s; // sizeof(short)=2
      int                i; // sizeof(double)=4
     };
//--- declare a simple structure instance
   CharShortInt ch_sh_in;
//--- display the size of each structure member
   Print("sizeof(ch_sh_in.c)=",sizeof(ch_sh_in.c));
   Print("sizeof(ch_sh_in.s)=",sizeof(ch_sh_in.s));
   Print("sizeof(ch_sh_in.i)=",sizeof(ch_sh_in.i));

//--- make sure the size of POD structure is equal to the sum of its members' size
   Print("sizeof(CharShortInt)=",sizeof(CharShortInt));
/*
  Result:
   sizeof(ch_sh_in.c)=1
   sizeof(ch_sh_in.s)=2
   sizeof(ch_sh_in.i)=4
   sizeof(CharShortInt)=8
*/
  }
```

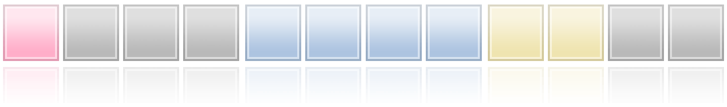
As we can see, the structure size is 8 and consists of the two 4-byte blocks. The first block contains the fields with 'char' and 'short' types, while the second one contains the field with 'int' type.



Now let's turn the first structure into the second one, which differs only in the field order, by moving the 'short' type member to the end.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- simple structure aligned to the 4-byte boundary
   struct CharIntShort pack(4)
     {
      char              c; // sizeof(char)=1
      int               i; // sizeof(double)=4
      short             s; // sizeof(short)=2
     };
//--- declare a simple structure instance
   CharIntShort ch_in_sh;
//--- display the size of each structure member
   Print("sizeof(ch_in_sh.c)=",sizeof(ch_in_sh.c));
   Print("sizeof(ch_in_sh.i)=",sizeof(ch_in_sh.i));
   Print("sizeof(ch_in_sh.s)=",sizeof(ch_in_sh.s));
//--- make sure the size of POD structure is equal to the sum of its members' size
   Print("sizeof(CharIntShort)=",sizeof(CharIntShort));
/*
  Result:
   sizeof(ch_in_sh.c)=1
   sizeof(ch_in_sh.i)=4
   sizeof(ch_in_sh.s)=2
   sizeof(CharIntShort)=12
*/
  }
```

Although the structure content has not changed, altering the member sequence has increased its size.



Alignment should also be considered when inheriting. Let's demonstrate this using the simple Parent structure having a single 'char' type member. The structure size without alignment is 1.

```
   struct Parent
     {
      char               c;    // sizeof(char)=1
     };
```

Let's create the Children child class featuring the 'short' (sizeof(short)=2) type member.

```
   struct Children pack(2) : Parent
     {
      short              s;   // sizeof(short)=2
     };
```

As a result, when setting alignment to 2 bytes, the structure size is equal to 4, although the size of its members is 3. In this example, 2 bytes are to be allocated to the Parent class, so that the access to the 'short' field of the child class is aligned to 2 bytes.

The knowledge of how memory is allocated for the structure members is necessary if an MQL5 application interacts with third-party data by writing/reading on the files or streams level.

The MQL5\Include\WinAPI directory of the Standard Library contains the functions for working with the WinAPI functions. These functions apply the structures with a specified alignment for the cases when it is required for working with WinAPI.

offsetof is a special command directly related to the pack attribute. It allows us to obtain a member offset from the beginning of the structure.

```
//--- declare the Children type variable
   Children child;
//--- detect offsets from the beginning of the structure
   Print("offsetof(Children,c)=",offsetof(Children,c));
   Print("offsetof(Children,s)=",offsetof(Children,s));
/*
  Result:
   offsetof(Children,c)=0
   offsetof(Children,s)=2
*/
```

Specifier 'final'

The use of the 'final' specifier during structure declaration prohibits further inheritance from this structure. If a structure requires no further modifications, or modifications are not allowed for security reasons, declare this structure with the 'final' modifier. In addition, all the members of the structure will also be implicitly considered final.

```
struct settings final
  {
  //--- Structure body
  };

struct trade_settings : public settings
  {
  //--- Structure body
  };
```

If you try to inherit from a structure with the 'final' modifier as shown in the above example, the compiler will return an error:

```
cannot inherit from 'settings' as it has been declared as 'final'
see declaration of 'settings'
```

Classes

Classes differ from structures in the following:

- the keyword class is used in declaration;
- by default, all class members have access specifier private, unless otherwise indicated. Data-members of the structure have the default type of access as public, unless otherwise indicated;
- class objects always have a table of virtual functions, even if there are no virtual functions declared in the class. Structures cannot have virtual functions;
- the new operator can be applied to class objects; this operator cannot be applied to structures;
- classes can be inherited only from classes, structures can be inherited only from structures.

Classes and structures can have an explicit constructor and destructor. If your constructor is explicitly defined, the initialization of a structure or class variable using the initializing sequence is impossible.

Example:

```
struct trade_settings
  {
   double take;         // values of the profit fixing price
   double stop;         // value of the protective stop price
   uchar  slippage;     // value of the acceptable slippage
   //--- Constructor
          trade_settings() { take=0.0; stop=0.0; slippage=5; }
   //--- Destructor
         ~trade_settings() { Print("This is the end"); }
  };
//--- Compiler will generate an error message that initialization is impossible
trade_settings my_set={0.0,0.0,5};
```

Constructors and Destructors

A constructor is a special function, which is called automatically when creating an object of a structure or class and is usually used to initialize class members. Further we will talk only about classes, while the same applies to structures, unless otherwise indicated. The name of a constructor must match the class name. The constructor has no return type (you can specify the void type).

Defined class members – strings, dynamic arrays and objects that require initialization – will be in any case initialized, regardless of whether there is a constructor.

Each class can have multiple constructors, differing by the number of parameters and the initialization list. A constructor that requires specifying parameters is called a parametric constructor.

A constructor with no parameters is called **a default constructor**. If no constructors are declared in a class, the compiler creates a default constructor during compilation.

```
//+------------------------------------------------------------------+
//| A class for working with a date                                  |
//+------------------------------------------------------------------+
class MyDateClass
  {
private:
   int               m_year;          // Year
   int               m_month;         // Month
   int               m_day;           // Day of the month
   int               m_hour;          // Hour in a day
   int               m_minute;        // Minutes
   int               m_second;        // Seconds
public:
   //--- Default constructor
                     MyDateClass(void);
   //--- Parametric constructor
                     MyDateClass(int h,int m,int s);
  };
```

A constructor can be declared in the class description and then its body can be defined. For example, two constructors of MyDateClass can be defined the following way:

```
//+------------------------------------------------------------------+
//| Default constructor                                              |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(void)
  {
//---
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=mdt.hour;
   m_minute=mdt.min;
   m_second=mdt.sec;
   Print(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Parametric constructor                                           |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(int h,int m,int s)
  {
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=h;
   m_minute=m;
   m_second=s;
   Print(__FUNCTION__);
  }
```

In the default constructor, all members of the class are filled using the TimeCurrent() function. In the parametric constructor only hour values are filled in. Other members of the class (m_year, m_month and m_day) will be automatically initialized with the current date.

The default constructor has a special purpose when initializing an array of objects of its class. The constructor, all parameters of which have default values, **is not** a default constructor. Here is an example:

```
//+------------------------------------------------------------------+
//| A class with a default constructor                               |
//+------------------------------------------------------------------+
class CFoo
  {
   datetime          m_call_time;     // Time of the last object call
public:
   //--- Constructor with a parameter that has a default value is not a default const
                     CFoo(const datetime t=0){m_call_time=t;};
   //--- Copy constructor
                     CFoo(const CFoo &foo){m_call_time=foo.m_call_time;};

   string ToString(){return(TimeToString(m_call_time,TIME_DATE|TIME_SECONDS));};
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
// CFoo foo; // This variant cannot be used - a default constructor is not set
//--- Possible options to create the CFoo object
```

```
   CFoo foo1(TimeCurrent());     // An explicit call of a parametric constructor
   CFoo foo2();                  // An explicit call of a parametric constructor with
   CFoo foo3=D'2009.09.09';      // An implicit call of a parametric constructor
   CFoo foo40(foo1);             // An explicit call of a copy constructor
   CFoo foo41=foo1;              // An implicit call of a copy constructor
   CFoo foo5;                    // An explicit call of a default constructor (if the
                                 // then a parametric constructor with a default valu
//--- Possible options to receive CFoo pointers
   CFoo *pfoo6=new CFoo();       // Dynamic creation of an object and receiving of a
   CFoo *pfoo7=new CFoo(TimeCurrent());// Another option of dynamic object creation
   CFoo *pfoo8=GetPointer(foo1); // Now pfoo8 points to object foo1
   CFoo *pfoo9=pfoo7;            // pfoo9 and pfoo7 point to one and the same object
   // CFoo foo_array[3];         // This option cannot be used - a default constructo
//--- Show the value of m_call_time
   Print("foo1.m_call_time=",foo1.ToString());
   Print("foo2.m_call_time=",foo2.ToString());
   Print("foo3.m_call_time=",foo3.ToString());
   Print("foo4.m_call_time=",foo4.ToString());
   Print("foo5.m_call_time=",foo5.ToString());
   Print("pfoo6.m_call_time=",pfoo6.ToString());
   Print("pfoo7.m_call_time=",pfoo7.ToString());
   Print("pfoo8.m_call_time=",pfoo8.ToString());
   Print("pfoo9.m_call_time=",pfoo9.ToString());
//--- Delete dynamically created arrays
   delete pfoo6;
   delete pfoo7;
   //delete pfoo8;  // You do not need to delete pfoo8 explicitly, since it points to
   //delete pfoo9;  // You do not need to delete pfoo9 explicitly. since it points to
  }
```

If you uncomment these strings

```
  //CFoo foo_array[3];     // This variant cannot be used - a default constructor is
```

or

```
  //CFoo foo_dyn_array[];  // This variant cannot be used - a default constructor is
```

then the compiler will return an error for them "default constructor is not defined".

If a class has a user-defined constructor, the default constructor is not generated by the compiler. This means that if a parametric constructor is declared in a class, but a default constructor is not declared, you can not declare the arrays of objects of this class. The compiler will return an error for this script:

```
//+------------------------------------------------------------------+
//| A class without a default constructor                            |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
```

```
                     CFoo(string name) { m_name=name;}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Get the "default constructor is not defined" error during compilation
   CFoo badFoo[5];
  }
```

In this example, the CFoo class has a declared parametric constructor - in such cases, the compiler does not create a default constructor automatically during compilation. At the same time when you declare an array of objects, it is assumed that all objects should be created and initialized automatically. During auto-initialization of an object, it is necessary to call a default constructor, but since the default constructor is not explicitly declared and not automatically generated by the compiler, it is impossible to create such an object. For this reason, the compiler generates an error at the compilation stage.

There is a special syntax to initialize an object using a constructor. Constructor initializers (special constructions for initialization) for the members of a struct or class can be specified in the initialization list.

An initialization list is a list of initializers separated by commas, which comes after the colon after the list of parameters of a constructor and precedes the body (goes before an opening brace). There are several requirements:

- Initialization lists can be used only in constructors;
- Parent members cannot be initialized in the initialization list;
- The initialization list must be followed by a definition (implementation) of a function.

Here is an example of several constructors for initializing class members.

```
//+------------------------------------------------------------------+
//| A class for storing the name of a character                      |
//+------------------------------------------------------------------+
class CPerson
  {
   string            m_first_name;     // First name
   string            m_second_name;    // Second name
public:
   //--- An empty default constructor
                     CPerson() {Print(__FUNCTION__);};
   //--- A parametric constructor
                     CPerson(string full_name);
   //--- A constructor with an initialization list
                     CPerson(string surname,string name): m_second_name(surname), m_fi
   void PrintName(){PrintFormat("Name=%s Surname=%s",m_first_name,m_second_name);};
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
CPerson::CPerson(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>=0)
     {
      m_first_name=StringSubstr(full_name,0,pos);
      m_second_name=StringSubstr(full_name,pos+1);
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Get an error "default constructor is not defined"
   CPerson people[5];
   CPerson Tom="Tom Sawyer";                       // Tom Sawyer
   CPerson Huck("Huckleberry","Finn");             // Huckleberry Finn
   CPerson *Pooh = new CPerson("Winnie","Pooh");  // Winnie the Pooh
   //--- Output values
   Tom.PrintName();
   Huck.PrintName();
   Pooh.PrintName();

   //--- Delete a dynamically created object
   delete Pooh;
  }
```

In this case, the CPerson class has three constructors:

1. An explicit default constructor, which allows creating an array of objects of this class;
2. A constructor with one parameter, which gets a full name as a parameter and divides it to the name and second name according to the found space;
3. A constructor with two parameters that contains an initialization list. Initializers - m_second_name(surname) and m_first_name(name).

Note that the initialization using a list has replaced an assignment. Individual members must be initialized as:

```
 class_member (a list of expressions)
```

In the initialization list, members can go in any order, but all members of the class will be initialized according to the order of their announcement. This means that in the third constructor, first the m_first_name member will be initialized, as it is announced first, and only after it m_second_name is initialized. This should be taken into account in cases where the initialization of some members of the class depends on the values in other class members.

If a default constructor is not declared in the base class, and at the same time one or more constructors with parameters are declared, you should always call one of the base class constructors in the initialization list. It goes through the comma as ordinary members of the list and will be called first during object initialization, no matter where in the initialization list it is located.

```
//+------------------------------------------------------------------+
//| Base class                                                       |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
   //--- A constructor with an initialization list
                     CFoo(string name) : m_name(name) { Print(m_name);}
  };
//+------------------------------------------------------------------+
//| Class derived from CFoo                                          |
//+------------------------------------------------------------------+
class CBar : CFoo
  {
   CFoo              m_member;      // A class member is an object of the parent
public:
   //--- A default constructor in the initialization list calls the constructor of a p
                     CBar(): m_member(_Symbol), CFoo("CBAR") {Print(__FUNCTION__);}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   CBar bar;
  }
```

In this example, when creating the bar object, a default constructor CBar() will be called, in which first a constructor for the parent CFoo is called, and then comes a constructor for the m_member class member.

A destructor is a special function that is called automatically when a class object is destroyed. The name of the destructor is written as a class name with a tilde (~). Strings, dynamic arrays and objects, requiring deinitialization, will be de-initialized anyway, regardless of the destructor presence or absence. If there is a destructor, these actions will be performed after calling the destructor.

Destructors are always virtual, regardless of whether they are declared with the virtual keyword or not.

Defining Class Methods

Class function-methods can be defined both inside the class and outside the class declaration. If the method is defined within a class, then its body comes right after the method declaration.

Example:

```
class CTetrisShape
  {
```

```
protected:
   int                  m_type;
   int                  m_xpos;
   int                  m_ypos;
   int                  m_xsize;
   int                  m_ysize;
   int                  m_prev_turn;
   int                  m_turn;
   int                  m_right_border;
public:
   void                 CTetrisShape();
   void                 SetRightBorder(int border) { m_right_border=border; }
   void                 SetYPos(int ypos)          { m_ypos=ypos;           }
   void                 SetXPos(int xpos)          { m_xpos=xpos;           }
   int                  GetYPos()                  { return(m_ypos);        }
   int                  GetXPos()                  { return(m_xpos);        }
   int                  GetYSize()                 { return(m_ysize);       }
   int                  GetXSize()                 { return(m_xsize);       }
   int                  GetType()                  { return(m_type);        }
   void                 Left()                     { m_xpos-=SHAPE_SIZE;    }
   void                 Right()                    { m_xpos+=SHAPE_SIZE;    }
   void                 Rotate()                   { m_prev_turn=m_turn; if(++m_turn>3) r
   virtual void         Draw()                     { return;                }
   virtual bool         CheckDown(int& pad_array[]);
   virtual bool         CheckLeft(int& side_row[]);
   virtual bool         CheckRight(int& side_row[]);
  };
```

Functions from SetRightBorder(int border) to Draw() are declared and defined directly inside the CTetrisShape class.

The CTetrisShape() constructor and methods CheckDown(int& pad_array[]), CheckLeft(int& side_row[]) and CheckRight(int& side_row[]) are only declared inside the class, but not defined yet. Definitions of these functions will be further in the code. In order to define the method outside the class, the scope resolution operator is used, the class name is used as the scope.

Example:

```
//+------------------------------------------------------------------+
//| Constructor of the basic class                                   |
//+------------------------------------------------------------------+
void CTetrisShape::CTetrisShape()
  {
   m_type=0;
   m_ypos=0;
   m_xpos=0;
   m_xsize=SHAPE_SIZE;
   m_ysize=SHAPE_SIZE;
   m_prev_turn=0;
   m_turn=0;
   m_right_border=0;
  }
//+------------------------------------------------------------------+
//| Checking ability to move down (for the stick and cube)           |
//+------------------------------------------------------------------+
bool CTetrisShape::CheckDown(int& pad_array[])
  {
   int i,xsize=m_xsize/SHAPE_SIZE;
//---
   for(i=0; i<xsize; i++)
     {
      if(m_ypos+m_ysize>=pad_array[i]) return(false);
     }
//---
   return(true);
  }
```

Public, Protected and Private Access Specifiers

When developing a new class, it is recommended to restrict access to the members from the outside. For this purpose keywords private or protected are used. In this case, hidden data can be accessed only from function-methods of the same class. If the *protected* keyword is used, hidden data can be accessed also from methods of classes - inheritors of this class. The same method can be used to restrict the access to functions-methods of a class.

If you need to completely open access to members and/or methods of a class, use the keyword public.

Example:

```
class CTetrisField
  {
private:
   int                 m_score;                            // Score
   int                 m_ypos;                             // Current position of the fi
   int                 m_field[FIELD_HEIGHT][FIELD_WIDTH]; // Matrix of the well
   int                 m_rows[FIELD_HEIGHT];               // Numbering of the well rows
   int                 m_last_row;                         // Last free row
   CTetrisShape       *m_shape;                            // Tetris figure
   bool                m_bover;                            // Game over
public:
   void                CTetrisField() { m_shape=NULL; m_bover=false; }
   void                Init();
   void                Deinit();
   void                Down();
   void                Left();
   void                Right();
   void                Rotate();
   void                Drop();
private:
   void                NewShape();
   void                 CheckAndDeleteRows();
   void                 LabelOver();
  };
```

Any class members and methods declared after the specifier **public:** (and before the next access specifier) are available in any reference to the class object by the program. In this example these are the following members: functions CTetrisField(), Init(), Deinit(), Down(), Left(), Right(), Rotate() and Drop().

Any members that are declared after the access specifier to the elements **private:** (and before the next access specifier) are available only to members-functions of this class. Specifiers of access to elements always end with a colon (:) and can appear in the class definition many times.

Any class members declared after the **protected:** access specifier (and up to the next access specifier) are available only to members-functions of this class and members-functions of the class descendants. When attempting to refer to the members featuring the private and protected specifiers from the outside, we get the compilation stage error. Example:

```
class A
  {
protected:
   //--- the copy operator is available only inside class A and its descendants
   void operator=(const A &)
     {
     }
  };
class B
  {
   //--- class A object declared
   A                 a;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- declare two B type variables
   B b1, b2;
   //--- attempt to copy one object into another
   b2=b1;
  }
```

When compiling the code, the error message is received — an attempt to call the remote copy operator:

```
attempting to reference deleted function 'void B::operator=(const B&)'   trash3.mq5
```

The second string below provides a more detailed description — the copy operator in class B was explicitly deleted, since the unavailable copy operator of class A is called:

```
   function 'void B::operator=(const B&)' was implicitly deleted because it invokes ir
```

Access to the members of the basis class can be redefined during inheritance in derived classes.

'delete' specifier

The delete specifier marks the class members-functions that cannot be used. This means if the program refers to such a function explicitly or implicitly, the error is received at the compilation stage already. For example, this specifier allows you to make parent methods unavailable in a child class. The same result can be achieved if we declare the function in the private area of the parent class (declarations in the private section). Here, using delete makes the code more readable and manageable at the level of descendants.

```
class A
  {
public:
                     A(void) {value=5;};
   double            GetValue(void) {return(value);}
private:
   double            value;
  };
class B: public A
  {
   double            GetValue(void)=delete;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the A type variable
   A a;
   Print("a.GetValue()=", a.GetValue());
//--- attempt to get value from the B type variable
   B b;
   Print("b.GetValue()=", b.GetValue()); // the compiler displays an error at this str
  }
```

The compiler message:

```
attempting to reference deleted function 'double B::GetValue()'
   function 'double B::GetValue()' was explicitly deleted here
```

The 'delete' specifier allows disabling auto casting or the copy constructor, which otherwise would have to be hidden in the private section as well. Example:

```
class A
  {
public:
   void              SetValue(double v) {value=v;}
   //--- disable int type call
   void              SetValue(int) = delete;
   //--- disable the copy operator
   void               operator=(const A&) = delete;
private:
   double             value;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare two A type variables
   A a1, a2;
   a1.SetValue(3);      // error!
   a1.SetValue(3.14);   // OK
   a2=a1;               // error!
  }
```

During the compilation, we get the error messages:

```
attempting to reference deleted function 'void A::SetValue(int)'
   function 'void A::SetValue(int)' was explicitly deleted here
attempting to reference deleted function 'void A::operator=(const A&)'
   function 'void A::operator=(const A&)' was explicitly deleted here
```

Specifier 'final'

The use of the 'final' specifier during class declaration prohibits further inheritance from this class. If the class interface requires no further modifications, or modifications are not allowed for security reasons, declare this class with the 'final' modifier. In addition, all the members of the class will also be implicitly considered final.

```
class CFoo final
  {
  //--- Class body
  };
 
class CBar : public CFoo
  {
  //--- Class body
  };
```

If you try to inherit form a class with the 'final' specifier as shown in the above example, the compiler will return an error:

```
cannot inherit from 'CFoo' as it has been declared as 'final'
see declaration of 'CFoo'
```

Unions (union)

Union is a special data type consisting of several variables sharing the same memory area. Therefore, the union provides the ability to interpret the same bit sequence in two (or more) different ways. Union declaration is similar to structure declaration and starts with the union keyword.

```
union LongDouble
```

```
{
  long   long_value;
  double double_value;
};
```

Unlike the structure, various union members belong to the same memory area. In this example, the union of LongDouble is declared with long and double type values sharing the same memory area. Please note that it is impossible to make the union store a *long* integer value and a *double* real value simultaneously (unlike a structure), since long_value and double_value variables overlap (in memory). On the other hand, an MQL5 program is able to process data containing in the union as an integer (long) or real (double) value at any time. Therefore, the union allows receiving two (or more) options for representing the same data sequence.

During the union declaration, the compiler automatically allocates the memory area sufficient to store the largest type (by volume) in the variable union. The same syntax is used for accessing the union element as for the structures – point operator.

```
union LongDouble
{
  long   long_value;
  double double_value;
};
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   LongDouble lb;
//--- get and display the invalid -nan(ind) number
   lb.double_value=MathArcsin(2.0);
   printf("1.  double=%f                integer=%I64X",lb.double_value,lb.long_value);
//--- largest normalized value (DBL_MAX)
   lb.long_value=0x7FEFFFFFFFFFFFFF;
   printf("2.  double=%.16e  integer=%I64X",lb.double_value,lb.long_value);
//--- smallest positive normalized (DBL_MIN)
   lb.long_value=0x0010000000000000;
   printf("3.  double=%.16e  integer=%.16I64X",lb.double_value,lb.long_value);
  }
/*  Execution result
    1.  double=-nan(ind)                 integer=FFF8000000000000
    2.  double=1.7976931348623157e+308   integer=7FEFFFFFFFFFFFFF
    3.  double=2.2250738585072014e-308   integer=0010000000000000
*/
```

Since the unions allow the program to interpret the same memory data in different ways, they are often used when an unusual type conversion is required.

The unions cannot be involved in the inheritance, and they also cannot have static members due to their very nature. In all other aspects, the *union* behaves like a structure with all its members having a zero offset. The following types cannot be the union members:

- dynamic arrays
- strings
- pointers to objects and functions
- class objects
- structure objects having constructors or destructors
- structure objects having members from the points 1-5

Similar to classes, the union is capable of having constructors and destructors, as well as methods. By default, the union members are of public access type. In order to create private elements, use the private keyword. All these possibilities are displayed in the example illustrating how to convert a color of the color type to ARGB as does the ColorToARGB() function.

```
//+------------------------------------------------------------------+
//| Union for color(BGR) conversion to ARGB                          |
//+------------------------------------------------------------------+
union ARGB
  {
   uchar             argb[4];
   color             clr;
   //--- constructors
                     ARGB(color col,uchar a=0){Color(col,a);};
                    ~ARGB(){};
   //--- public methods
public:
   uchar   Alpha(){return(argb[3]);};
   void    Alpha(const uchar alpha){argb[3]=alpha;};
   color   Color(){ return(color(clr));};
   //--- private methods
private:
   //+------------------------------------------------------------------+
   //| set the alpha channel value and color                            |
   //+------------------------------------------------------------------+
   void    Color(color col,uchar alpha)
     {
      //--- set color to clr member
      clr=col;
      //--- set the Alpha component value - opacity level
      argb[3]=alpha;
      //--- interchange the bytes of R and B components (Red and Blue)
      uchar t=argb[0];argb[0]=argb[2];argb[2]=t;
     };
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- 0x55 means 55/255=21.6 % (0% - fully transparent)
```

```
   uchar alpha=0x55;
//--- color type is represented as 0x00BBGGRR
   color test_color=clrDarkOrange;
//--- values of bytes from the ARGB union are accepted here
   uchar argb[];
   PrintFormat("0x%.8X - here is how the 'color' type look like for %s, BGR=(%s)",
               test_color,ColorToString(test_color,true),ColorToString(test_color));
//--- ARGB type is represented as 0x00RRGGBB, RR and BB components are swapped
   ARGB argb_color(test_color);
//--- copy the bytes array
   ArrayCopy(argb,argb_color.argb);
//--- here is how it looks in ARGB representation
   PrintFormat("0x%.8X - ARGB representation with the alpha channel=0x%.2x, ARGB=(%d,
               argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- add opacity level
   argb_color.Alpha(alpha);
//--- try defining ARGB as 'color' type
   Print("ARGB as color=(",argb_color.clr,")  alpha channel=",argb_color.Alpha());
//--- copy the bytes array
   ArrayCopy(argb,argb_color.argb);
//--- here is how it looks in ARGB representation
   PrintFormat("0x%.8X - ARGB representation with the alpha channel=0x%.2x, ARGB=(%d,
               argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- check with the ColorToARGB() function results
   PrintFormat("0x%.8X - result of ColorToARGB(%s,0x%.2x)",ColorToARGB(test_color,alph
               ColorToString(test_color,true),alpha);
  }
/* Execution result
   0x00008CFF - here is how the color type looks for clrDarkOrange, BGR=(255,140,0)
   0x00FF8C00 - ARGB representation with the alpha channel=0x00, ARGB=(0,255,140,0)
   ARGB as color=(0,140,255)  alpha channel=85
   0x55FF8C00 - ARGB representation with the alpha channel=0x55, ARGB=(85,255,140,0)
   0x55FF8C00 - result of ColorToARGB(clrDarkOrange,0x55)
*/
```

Interfaces

An interface allows determining specific functionality, which a class can then implement. In fact, an interface is a class that cannot contain any members, and may not have a constructor and/or a destructor. All methods declared in an interface are purely virtual, even without an explicit definition.

An interface is defined using the "interface" keyword. Example:

```
//--- Basic interface for describing animals
interface IAnimal
  {
//--- The methods of the interface have public access by default
   void Sound();  // The sound produced by the animal
  };
```

```
//+------------------------------------------------------------------+
//|  The CCat class is inherited from the IAnimal interface          |
//+------------------------------------------------------------------+
class CCat : public IAnimal
  {
public:
                     CCat() { Print("Cat was born"); }
                    ~CCat() { Print("Cat is dead");  }
   //--- Implementing the Sound method of the IAnimal interface
   void Sound(){ Print("meou"); }
  };
//+------------------------------------------------------------------+
//|  The CDog class is inherited from the IAnimal interface          |
//+------------------------------------------------------------------+
class CDog : public IAnimal
  {
public:
                     CDog() { Print("Dog was born"); }
                    ~CDog() { Print("Dog is dead");  }
   //--- Implementing the Sound method of the IAnimal interface
   void Sound(){ Print("guaf"); }
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- An array of pointers to objects of the IAnimal type
   IAnimal *animals[2];
//--- Creating child classes of IAnimal and saving pointers to them into an array
   animals[0]=new CCat;
   animals[1]=new CDog;
//--- Calling the Sound() method of the basic IAnimal interface for each child
   for(int i=0;i<ArraySize(animals);++i)
      animals[i].Sound();
//--- Deleting objects
   for(int i=0;i<ArraySize(animals);++i)
      delete animals[i];
//--- Execution result
/*
   Cat was born
   Dog was born
   meou
   guaf
   Cat is dead
   Dog is dead
*/
  }
```

Like with abstract classes, an interface object cannot be created without inheritance. An interface can only be inherited from other interfaces and can be a parent for a class. An interface is always publicly visible.

An interface cannot be declared within a class or structure declaration, but a pointer to the interface can be saved in a variable of type void *. Generally speaking, a pointer to an object of any class can be saved into a variable of type void *. In order to convert a void * pointer to a pointer to an object of a particular class, use the dynamic_cast operator. If conversion is not possible, the result of the dynamic_cast operation will be NULL.

See also

Object-Oriented Programming

Dynamic Array Object

Dynamic Arrays

Maximum 4-dimension array can be declared. When declaring a dynamic array (an array of unspecified value in the first pair of square brackets), the compiler automatically creates a variable of the above structure (a dynamic array object) and provides a code for the correct initialization.

Dynamic arrays are automatically freed when going beyond the visibility area of the block they are declared in.

Example:

```
double matrix[][10][20]; // 3-dimensional dynamic array
ArrayResize(matrix,5);    // Set the size of the first dimension
```

Static Arrays

When all significant array dimensions are explicitly specified, the compiler pre-allocates the necessary memory size. Such an array is called static. Nevertheless, the compiler allocates additional memory for the object of a dynamic array, which (object) is associated with the pre-allocated static buffer (memory part for storing the array).

Creating a dynamic array object is due to the possible need to pass this static array as a parameter to some function.

Examples:

```
double stat_array[5]; // 1-dimensional static array
some_function(stat_array);
...
bool some_function(double& array[])
  {
   if(ArrayResize(array,100)<0) return(false);
   ...
   return(true);
  }
```

Arrays in Structures

When a static array is declared as a member of a structure, a dynamic array object is not created. This is done to ensure compatibility of data structures used in the Windows API.

However, static arrays that are declared as members of structures can also be passed to MQL5 functions. In this case, when passing the parameter, a temporary object of a dynamic array will be created. Such an object is linked with the static array - member of structure.

See also

Array Functions, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Matrices and vectors

Type vector is a special data type in MQL5, which enables operations with vectors. A vector is a one-dimensional array of type double. It is one of the fundamental concepts of linear algebra, which is used in many fields of science, including physics, geometry, and others. Vectors are used to solve systems of linear equations, in 3D graphics and in other application areas. Vectors can be added and multiplied. The length or distance between vectors can be obtained through the Norm. In programming, vectors are usually represented by arrays of homogeneous elements, which may have no regular vector operations, i.e. when arrays cannot be added or multiply, and they have no norm.

Vectors can be represented as row vectors and string vectors when working with matrices. Also, vectors in linear algebra use the concepts of covariance and contravariance. These concepts do not make any difference when writing an MQL5 code, as only the programmer decides what each object of the vector type is. For example, it can be rotation, displacement or compression vector in 3D graphics.

Generally speaking, from the point of view of linear algebra, a number is also a vector, but in a one-dimensional vector space. A vector itself can be considered as a special case of a matrix.

Type matrix is another special data type in MQL5 to represent matrices. A matrix is actually a two-dimensional array of type double. Vectors and matrices have been introduced into MQL5 for easier operations with certain types of data sets. With them, developers can benefit from the linear algebra possibilities in a simple and math-like form. Matrices can be used to compactly write systems of linear or differential equations. The number of matrix rows corresponds to the number of equations, while the the number of columns is equal to the number of unknowns. As a result, systems of linear equations can be solved through matrix operations.

The following data types exist:

- matrix — a matrix containing double elements.
- matrixf — a matrix containing float elements.
- matrixc — a matrix containing complex elements.
- vector — a vector containing double elements.
- vectorf — a vector containing float elements.
- vectorc — a vector containing complex elements.

Template functions support notations like matrix<double>, matrix<float>, vector<double>, vector<float> instead of the corresponding types.

The following algebraic operations are defined for the matrices:

- Addition of same-size matrices
- Multiplication of suitable-size matrices: the number of columns in the left matrix must equal the number of rows in the right matrix
- Matrix multiplication by a column vector; multiplication of a row vector by a matrix according to the matrix multiplication rule. In this sense the vector is a special case of a matrix
- Matrix multiplication by a number, that is, by a scalar

Mathematics considers many different matrix types. For example, identity matrix, symmetric, skew-symmetric, upper and lower triangular matrices, and other types. Various Normal forms play an important role in the matrix theory. They represent a certain canonical form of a matrix, which can be obtained by means of certain transformations. In practice, normal forms that have additional properties, such as for example stability, are used.

The use of vectors and matrices, or rather, of special methods of the relevant types, enables the creation of simpler, briefer and clearer code, which is close to mathematical notation. With these methods, you can avoid the need to create nested loops or to mind correct indexing of arrays in calculations. Therefore, the use of these methods increases reliability and speed in developing complex programs.

List of matrix and vector methods

Types matrix and vector include methods that correspond to the relevant NumPy library methods. Using these methods, you can translate algorithms and codes from Python to MQL5 with minimum efforts. A lot of data processing tasks, mathematical equations, neural networks and machine learning tasks can be solved using ready-made Python methods and libraries.

Method matrix/vector	Analogous method in NumPy	Description
void matrix.Eye(const int rows, const int cols, const int ndiag=0)	eye	Construct a matrix with ones on the diagonal and zeros elsewhere
void matrix.Identity(const int rows)	identity	Construct a square matrix with ones on the main diagonal
void matrix.Ones(const int rows, const int cols)	ones	Construct a new matrix of given rows and columns, filled with ones
void matrix.Zeros(const int rows, const int cols)	zeros	Construct a new matrix of given rows and columns, filled with zeros
void matrix.Full(const int rows, const int cols, const scalar value)	full	Construct a new matrix of given rows and columns, filled with scalar value
void matrix.Copy(const matrix a)	copy	Construct a copy of the given matrix
void matrix.FromBuffer(const int rows, const int cols, const scalar array[], const int count=-1, const int offset=0)	frombuffer	Construct a matrix created from a 1-dimensional array
void matrix.FromFile(const int rows, condt int cols, const int file_handle, const int count=-1, const int offset=0)	fromfile	Construct a matrix from data in a text or binary file
void vector.FromString(const string source, const string sep=" ")	fromstring	Construct a vector initialized from text data in a string
void vector.Arange(const scalar start, const scalar stop, const scalar step=1)	arange	Construct evenly spaced values within a given interval

Method matrix/vector	Analogous method in NumPy	Description
void matrix.Diag(const vector v, const int ndiag=0)	diag	Extract a diagonal or construct a diagonal matrix
void matrix.Tri(const int rows, const int cols, const int ndiag=0)	tri	Construct a matrix with ones at and below the given diagonal and zeros elsewhere
void matrix.Tril(const int rows, const int cols, const scalar array[], const int ndiag=0)	tril	Return a copy of a matrix with elements above the k-th diagonal zeroed
void matrix.Triu(const int rows, const int cols, const scalar array[], const int ndiag=0)	triu	Return a copy of a matrix with the elements below the k-th diagonal zeroed
void matrix.Vander(const vector v, const int cols=-1, const bool increasing=false)	vander	Generate a Vandermonde matrix
vector matrix.Row(const unsigned nrow)		Return a row vector
vector matrix.Col(const unsigned ncol)		Return a column vector
unsigned matrix.Rows()		Return the number of rows in a matrix
unsigned matrix.Cols()		Return the number of columns in a matrix
void matrix.Init()		Initialize a matrix
matrix matrix.Transpose()	transpose	Reverse or permute the axes of a matrix; returns the modified matrix
matrix matrix.Dot(const matrix b)	dot	Dot product of two matrices
matrix matrix.Inner(const matrix b)	inner	Inner product of two matrices
matrix matrix.Outer(const matrix b)	outer	Compute the outer product of two matrices
matrix matrix.MatMul(const matrix b)	matmul	Matrix product of two matrices
matrix matrix.MatrixPower(const int power)	matrix_power	Raise a square matrix to the (integer) power n
matrix matrix.Kron(const matrix b)	kron	Return Kronecker product of two matrices
bool matrix.Cholesky(matrix& L)	cholesky	Return the Cholesky decomposition

Method matrix/vector	Analogous method in NumPy	Description
bool matrix.QR(matrix& Q, matrix& R)	qr	Compute the qr factorization of a matrix
bool matrix.SVD(matrix& U, matrix& V, vector& singular_values)	svd	Singular value decomposition
bool matrix.Eig(matrix& eigen_vectors, vector& eigen_values)	eig	Compute the eigenvalues and right eigenvectors of a square matrix
bool matrix.EigH(matrix& eigen_vectors, vector& eigen_values)	eigh	Return the eigenvalues and eigenvectors of a Hermitian matrix
bool matrix.EigVals(vector& eigen_values)	eigvals	Compute the eigenvalues of a general matrix
bool matrix.EigValsH(vector& eigen_values)	eigvalsh	Compute the eigenvalues of a Hermitian matrix
bool matrix.LU(matrix& L, matrix& U)		LU decomposition of a matrix as the product of a lower triangular matrix and an upper triangular matrix
bool matrix.LUP(matrix& L, matrix& U, matrix& P)		LUP decomposition with partial pivoting, which refers to LU decomposition with row permutations only: PA=LU
double matrix.Norm(const norm)	norm	Return matrix or vector norm
double matrix.Cond(const norm)	cond	Compute the condition number of a matrix
vector matrix.Spectrum()		Compute spectrum of a matrix as the set of its eigenvalues from the product AT*A
double matrix.Det()	det	Compute the determinant of an array
int matrix.Rank()	matrix_rank	Return matrix rank of array using the Gaussian method
int matrix.SLogDet(int& sign)	slogdet	Compute the sign and logarithm of the determinant of an array
double matrix.Trace()	trace	Return the sum along diagonals of the matrix
vector matrix.Solve(const vector b)	solve	Solve a linear matrix equation, or system of linear algebraic equations

Method matrix/vector	Analogous method in NumPy	Description
vector matrix.LstSq(const vector b)	lstsq	Return the least-squares solution of linear algebraic equations (for non-square or degenerate matrices)
matrix matrix.Inv()	inv	Compute the (multiplicative) inverse of a matrix
matrix matrix.PInv()	pinv	Compute the pseudo-inverse of a matrix by the Moore-Penrose method
int matrix.Compare(const matrix matrix_c, const double epsilon) int matrix.Compare(const matrix matrix_c, const int digits) int vector.Compare(const vector vector_c, const double epsilon) int vector.Compare(const vector vector_c, const int digits)		Compare the elements of two matrices/vectors with the specified precision
double matrix.Flat(const ulong index) bool matrix.Flat(const ulong index,const double value)	flat	Allows addressing a matrix element through one index instead of two
double vector.ArgMax() double matrix.ArgMax() vector matrix.ArgMax(const int axis)	argmax	Return the index of the maximum value
double vector.ArgMin() double matrix.ArgMin() vector matrix.ArgMin(const int axis)	argmin	Return the index of the minimum value
double vector.Max() double matrix.Max() vector matrix.Max(const int axis)	max	Return the maximum value in a matrix/vector
double vector.Mean() double matrix.Mean() vector matrix.Mean(const int axis)	mean	Compute the arithmetic mean of element values
double vector.Min() double matrix.Min() vector matrix.Min(const int axis)	min	Return the minimum value in a matrix/vector
double vector.Sum() double matrix.Sum() vector matrix.Sum(const int axis)	sum	Return the sum of the matrix/vector elements which can also be performed for the given axis (axes).
void vector.Clip(const double min_value,const double max_value)	clip	Limit the elements of a matrix/vector to a specified range

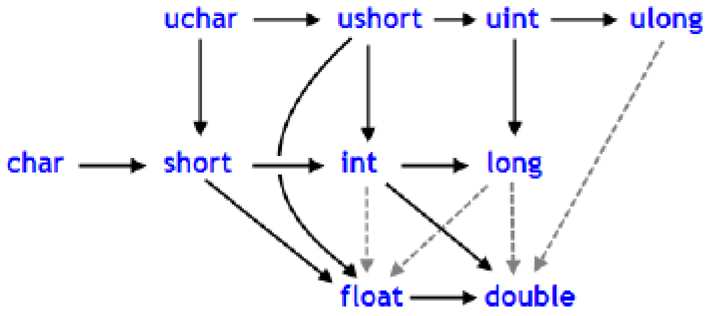
Method matrix/vector	Analogous method in NumPy	Description
void matrix.Clip(const double min_value,const double max_value)		of valid values
vector vector.CumProd() vector matrix.CumProd() matrix matrix.CumProd(const int axis)	cumprod	Return the cumulative product of matrix/vector elements, including those along the given axis
vector vector.CumSum() vector matrix.CumSum() matrix matrix.CumSum(const int axis)	cumsum	Return the cumulative sum of matrix/vector elements, including those along the given axis
double vector.Prod(const double initial=1) double matrix.Prod(const double initial=1) vector matrix.Prod(const int axis,const double initial=1)	prod	Return the product of the matrix/vector elements which can also be performed for the given axis
void matrix.Reshape(const ulong rows, const ulong cols)	reshape	Change the shape of a matrix without changing its data
void matrix.Resize(const ulong rows,const ulong cols)	resize	Return a new matrix with a changed shape and size
bool matrix.SwapRows(const ulong row1, const ulong row2)		Swap rows in a matrix
bool matrix.SwapCols(const ulong col1, const ulong col2)		Swap columns in a matrix
double vector.Ptp() double matrix.Ptp() vector matrix.Ptp(const int axis)	ptp	Return the range of values of a matrix/vector or of the given matrix axis, equivalent to Max() - Min()
double vector.Percentile(const int percent) double matrix.Percentile(const int percent) vector matrix.Percentile(const int percent,const int axis)	percentile	Return the specified percentile of values of matrix/vector elements or elements along the specified axis. Valid values of the 'percent' parameter are in the range [0, 100]
double vector.Quantile(const int percent) double matrix.Quantile(const int percent) vector matrix.Quantile(const int percent,const int axis)	quantile	Return the specified quantile of values of matrix/vector elements or elements along the specified axis. The 'percent' parameter takes values in the range [0, 1]
double vector.Median() double matrix.Median()	median	Compute the median of the matrix/vector elements. The

Method matrix/vector	Analogous method in NumPy	Description
vector matrix.Median(const int axis)		median is the middle value that separates the highest half of the array/vector elements from the lowest half of elements.
double vector.Average() double matrix.Average() vector matrix.Average(const int axis)	average	Compute the arithmetic mean of matrix/vector values. The sum of the weights in the denominator cannot be equal to 0, but some weights can be 0
double vector.Std() double matrix.Std() vector matrix.Std(const int axis)	std	Return the standard deviation of values of matrix/vector elements or of elements along the given axis
double vector.Var() double matrix.Var() vector matrix.Var(const int axis)	var	Compute the variance of values of matrix/vector elements
double vector.CorrCoef(const vector& v) matrix matrix.CorrCoef()	corrcoef	Compute the Pearson correlation coefficient (linear correlation coefficient). The correlation coefficient is in the range [-1, 1]
vector vector.Correlate(const vector& v,enum mode)	correlate	Compute the cross-correlation of two vectors. The 'mode' parameter determines the linear convolution calculation mode
vector vector.Convolve(const vector& v, enum mode)	convolve	Return the discrete, linear convolution of two vectors The 'mode' parameter determines the linear convolution calculation mode
matrix matrix.Cov() matrix vector.Cov(const vector& v); (resulting matrix 2 x 2)	cov	Compute the covariance matrix. The covariance of two samples (two random variables) is a measure of their linear dependence

Typecasting

Casting Numeric Types

Often a necessity occurs to convert one numeric type into another. Not all numeric types can be converted into another. Here is the scheme of allowed casting:



Solid lines with arrows indicate changes that are performed almost without any loss of information. Instead of the char type, the bool type can be used (both take 1 byte of memory), instead of type int, the color type can be used (4 bytes), instead of the long type, datetime can be used (take 8 bytes). The four dashed grey lines, also arrowed, denote conversions, when the loss of precision can occur. For example, the number of digits in an integer equal to 123456789 (int) is higher than the number of digits that can be represented by float.

```
   int n=123456789;
   float f=n;    // the content of f is equal to 1.234567892E8
   Print("n = ",n,"   f = ",f);
   // result n= 123456789    f= 123456792.00000
```

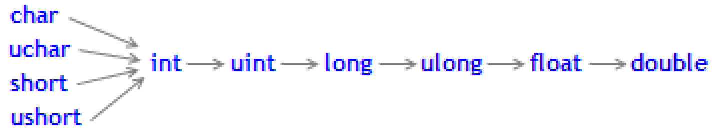
A number converted into float has the same order, but is less accurate. Conversions, contrary to black arrows, can be performed with possible data loss. Conversions between char and uchar, short and ushort, int and uint, long and ulong (conversions to both sides), may lead to the loss of data.

As a result of converting floating point values to integer type, the fractional part is always deleted. If you want to round off a float to the nearest whole number (which in many cases is more useful), you should use MathRound().

Example:

```
//--- Gravitational acceleration
   double g=9.8;
   double round_g=(int)g;
   double math_round_g=MathRound(g);
   Print("round_g = ",round_g);
   Print("math_round_g = ",math_round_g);
/*
   Result:
   round_g = 9
   math_round_g = 10
*/
```

If two values are combined by a binary operator, before the operation execution the operand of a lower type is converted to the higher type in accordance with the priority given in the below scheme:



The data types char, uchar, short, and ushort unconditionally are converted to the int type.

Examples:

```
   char    c1=3;
//--- First example
   double d2=c1/2+0.3;
   Print("c1/2 + 0.3 = ",d2);
// Result:   c1/2+0.3 = 1.3

//--- Second example
   d2=c1/2.0+0.3;
   Print("c1/2.0 + 0.3 = ",d2);
// Result:   c1/2.0+0.3 = 1.8
```

The calculated expression consists of two operations. In the first example, the variable c1 of the char type is converted to a temporary variable of the int type, because the second operand in the division operation, the constant 2, is of the higher type int. As a result of the integer division 3/2 we get the value 1, which is of the int type.

In the second operation of the first example, the second operand is the constant 0.3, which is of the double type, so the result of the first operation is converted into a temporary variable of the double type with a value of 1.0.

In the second example the variable of the char type c1 is converted to a temporary variable of the double type, because the second operand in the division operation, the constant 2.0, is of the double type; no further conversions are made.

Typecasting of Numeric Types

In the expressions of the MQL5 language both explicit and implicit typecasting can be used. The explicit typecasting is written as follows:

```
var_1 = (type)var_2;
```

An expression or function execution result can be used as the var_2 variable. The function style notation of the explicit typecasting is also possible:

```
var_1 = type(var_2);
```

Let's consider an explicit typecasting on the basis of the first example.

```
//--- Third example
   double d2=(double)c1/2+0.3;
   Print("(double)c1/2 + 0.3 = ",d2);
// Result:   (double)c1/2+0.3 = 1.80000000
```

Before the division operation is performed, the c1 variable is explicitly cast to the double type. Now the integer constant 2 is cast to the value 2.0 of the double type, because as a result of converting the first operand has taken the double type. In fact, the explicit typecasting is a unary operation.

Besides, when trying to cast types, the result may go beyond the permissible range. In this case, the truncation occurs. For example:

```
   char c;
   uchar u;
   c=400;
   u=400;
   Print("c = ",c); // Result c=-112
   Print("u = ",u); // Result u=144
```

Before operations (except for the assignment ones) are performed, the data are converted into the maximum priority type. Before assignment operations are performed, the data are cast into the target type.

Examples:

```
   int    i=1/2;        // no types casting, the result is 0
   Print("i = 1/2  ",i);

   int k=1/2.0;         // the expression is cast to the double type,
   Print("k = 1/2  ",k);  // then is to the target type of int, the result is 0

   double d=1.0/2.0;    // no types casting, the result is 0.5
   Print("d = 1/2.0; ",d);

   double e=1/2.0;      // the expression is cast to the double type,
   Print("e = 1/2.0; ",e);// that is the same as the target type, the result is 0.5

   double x=1/2;        // the expression of the int type is cast to the double target
   Print("x = 1/2; ",x);  // the result is 0.0
```

When converting long/ulong type into double, precision may be lost in case the integer value is greater than 9223372036854774784 or less than -9223372036854774784.

```
void OnStart()
  {
   long l_max=LONG_MAX;
   long l_min=LONG_MIN+1;
//--- define the highest integer value, which does not lose accuracy when being cast t
   while(l_max!=long((double)l_max))
      l_max--;
//--- define the lowest integer value, which does not lose accuracy when being cast to
   while(l_min!=long((double)l_min))
      l_min++;
//--- derive the found interval for integer values
```

```
   PrintFormat("When casting an integer value to double, it must be "
               "within [%I64d, %I64d] interval",l_min,l_max);
//--- now, let's see what happens if the value falls out of this interval
   PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))=%I64d",
               l_max+1,double(l_max+1),long(double(l_max+1)));
   PrintFormat("l_min-1=%I64d, double(l_min-1)=%.f, ulong(double(l_min-1))=%I64d",
               l_min-1,double(l_min-1),long(double(l_min-1)));
//--- receive the following result
// When casting an integer value to double, it should be within [-9223372036854774784,
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong(double(l_m
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulong(double(l
  }
```

Typecasting for the String Type

The string type has the highest priority among simple types. Therefore, if one of operands of an operation is of the string type, the second operand will be cast to a string automatically. Note that for a string, a single dyadic two-place operation of addition is possible. The explicit casting of string to any numeric type is allowed.

Examples:

```
   string s1=1.0/8;            // the expression is cast to the double type,
   Print("s1 = 1.0/8; ",s1);     //  then is to the target type of string,
// result is "0.12500000" (a string containing 10 characters)

   string s2=NULL;             // string deinitialization
   Print("s2 = NULL; ",s2);      // the result is an empty string
   string s3="Ticket N"+12345; // the expression is cast to the string type
   Print("s3 = \"Ticket N\"+12345",s3);

   string str1="true";
   string str2="0,255,0";
   string str3="2009.06.01";
   string str4="1.2345e2";
   Print(bool(str1));
   Print(color(str2));
   Print(datetime(str3));
   Print(double(str4));
```

Typecasting of Base Class Pointers to Pointers of Derivative Classes

Objects of the open generated class can also be viewed as objects of the corresponding base class. This leads to some interesting consequences. For example, despite the fact that objects of different classes, generated by a single base class, may differ significantly from each other, we can create a linked list (List) of them, as we view them as objects of the base type. But the converse is not true: the base class objects are not automatically objects of a derived class.

You can use the explicit casting to convert the base class pointers to the pointers of a derived class. But you must be fully confident in the admissibility of such a transformation, because otherwise a critical runtime error will occur and the mql5 program will be stopped.

Dynamic typecasting using dynamic_cast operator

Dynamic typecasting is performed using dynamic_cast operator that can be applied only to pointers to classes. Type validation is performed at runtime. This means that the compiler does not check the data type applied for typecasting when dynamic_cast operator is used. If a pointer is converted to a data type which is not the actual type of an object, the result is NULL.

```
dynamic_cast <type-id> ( expression )
```

The *type-id* parameter in angle brackets should point to a previously defined class type. Unlike C++, *expression* operand type can be of any value except for void.

Example:

```
class CBar { };
class CFoo : public CBar { };

void OnStart()
  {
   CBar bar;
//--- dynamic casting of *bar pointer type to *foo pointer is allowed
   CFoo *foo = dynamic_cast<CFoo *>(&bar); // no critical error
   Print(foo);                             // foo=NULL
//--- an attempt to explicitly cast a Bar type object reference to a Foo type object
   foo=(CFoo *)&bar;                       // critical runtime error
   Print(foo);                             // this string is not executed
  }
```

See also

Data Types

Void Type and NULL Constant

Syntactically the void type is a fundamental type along with types of char, uchar, bool, short, ushort, int, uint, color, long, ulong, datetime, float, double and string. This type is used either to indicate that the function does not return any value, or as a function parameter it denotes the absence of parameters.

The predefined constant variable **NULL** is of the *void* type. It can be assigned to variables of any other fundamental types without conversion. The comparison of fundamental type variables with the **NULL** value is allowed.

Example:

```
//--- If the string is not initialized, then assign our predefined value to it
if(some_string==NULL) some_string="empty";
```

Also **NULL** can be compared to pointers to objects created with the new operator.

See also

Variables, Functions

User-defined types

The typedef keyword in C++ allows creating user-defined data types. To do this, simply specify a new data type name for an already existing data type. The new data type is not created. A new name for the existing type is defined instead. User-defined types make applications more flexible: sometimes, it is enough to change typedef instructions using substitution macros (#define). User-defined types also improve code readability since it is possible to apply custom names to standard data types using typedef. The general format of the entry for creating a user-defined type:

```
   typedef type new_name;
```

Here, *type* means any acceptable data type, while *new_name* is a new name of the type. A new name is set only as an addition (not as a replacement) to an existing type name. MQL5 allows creating pointers to functions using typedef.

Pointer to the function

A pointer to a function is generally defined in the following format

```
   typedef function_result_type (*Function_name_type)(list_of_input_parameters_types);
```

where after typedef, the function signature (number and type of input parameters, as well as a type of a result returned by the function) is set. Below is a simple example of creating and applying a pointer to a function:

```
//--- declare a pointer to a function that accepts two int parameters
   typedef int (*TFunc)(int,int);
//--- TFunc is a type, and it is possible to declare the variable pointer to the funct
   TFunc func_ptr; // pointer to the function
//--- declare the functions corresponding to the TFunc description
   int sub(int x,int y) { return(x-y); }  // subtract one number from another
   int add(int x,int y) { return(x+y); }  // addition of two numbers
   int neg(int x)       { return(~x);  }  // invert bits in the variable
//--- the func_ptr variable may store the function address to declare it later
   func_ptr=sub;
   Print(func_ptr(10,5));
   func_ptr=add;
   Print(func_ptr(10,5));
   func_ptr=neg;           // error: neg does not have int (int,int) type
   Print(func_ptr(10));    // error: two parameters needed
```

In this example, the *func_ptr* variable may receive the *sub* and *add* functions since they have two inputs each of int type as defined in the *TFunc* pointer to the function. On the contrary, the *neg* function cannot be assigned to the func_ptr pointer since its signature is different.

Arranging event models in the user interface

Pointers to functions allow you to easily create processing of events when creating a user interface. Let's use an example from the CButton section to show how to create buttons and add the functions for handling pressing to them. First, define a pointer to the *TAction* function to be called by pressing the button and create three functions according to the *TAction description*.

```
//--- create a custom function type
typedef int(*TAction)(string,int);
//+------------------------------------------------------------------+
//|  Open the file                                                  |
//+------------------------------------------------------------------+
int Open(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(1);
  }
//+------------------------------------------------------------------+
//|  Save the file                                                  |
//+------------------------------------------------------------------+
int Save(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(2);
  }
//+------------------------------------------------------------------+
//|  Close the file                                                  |
//+------------------------------------------------------------------+
int Close(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(3);
  }
```

Then, create the MyButton class from CButton, where we should add the *TAction* pointer to the function.

```
//+------------------------------------------------------------------+
//| Create the button class with the events processing function      |
//+------------------------------------------------------------------+
class MyButton: public CButton
  {
private:
   TAction           m_action;                    // chart events handler
public:
                     MyButton(void){}
                    ~MyButton(void){}
   //--- constructor specifying the button text and the pointer to the events handlin
                     MyButton(string text, TAction act)
     {
      Text(text);
      m_action=act;
     }
   //--- set the custom function called from the OnEvent() events handler
   void              SetAction(TAction act){m_action=act;}
   //--- standard chart events handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
     {
      if(m_action!=NULL && lparam==Id())
        {
         //--- call the custom m_action() handler
         m_action(sparam,(int)lparam);
         return(true);
        }
      else
      //--- return the result of calling the handler from the CButton parent class
         return(CButton::OnEvent(id,lparam,dparam,sparam));
     }
  };
```

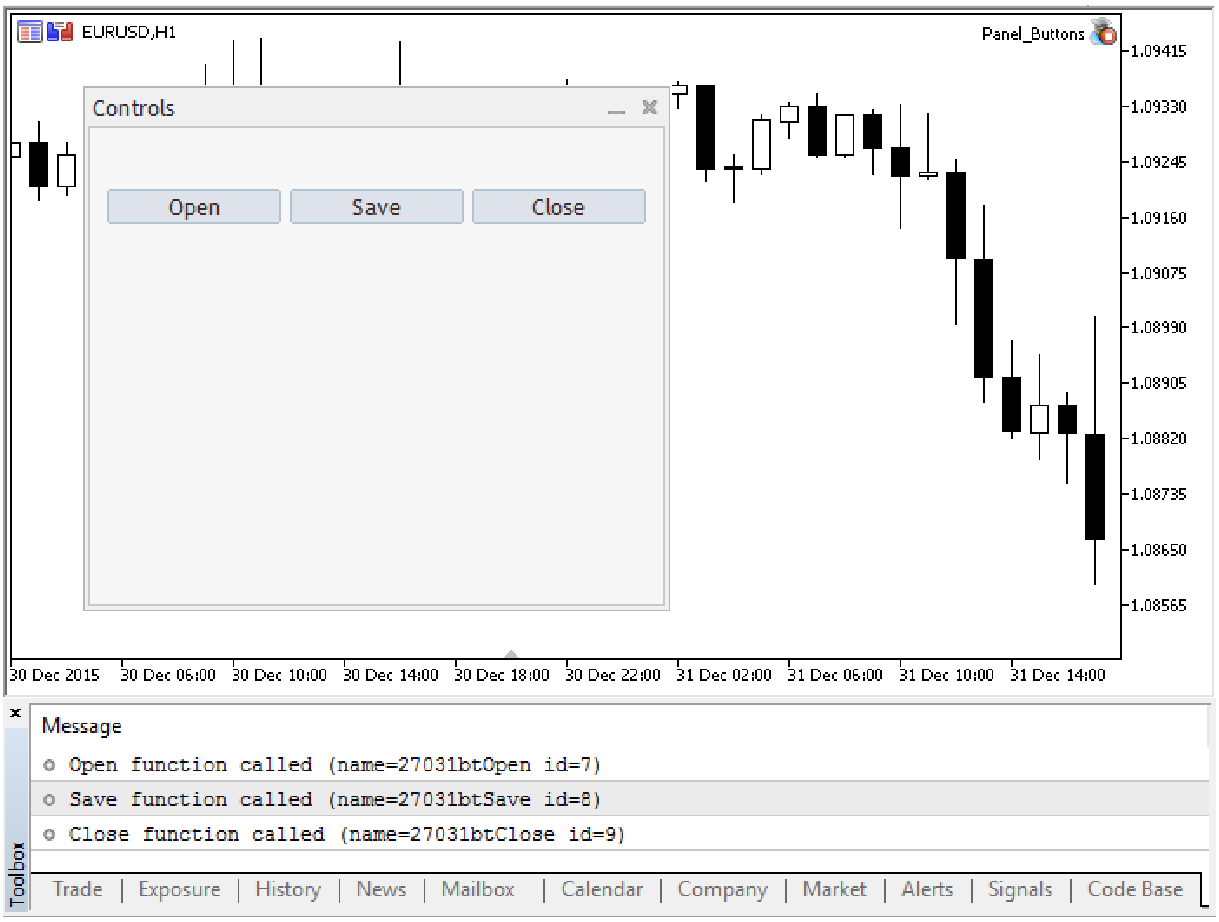
Create the CControlsDialog derivative class from CAppDialog, add the *m_buttons* array to it for storing the buttons of the *MyButton* type, as well as the *AddButton(MyButton &button)* and *CreateButtons() methods*.

```
//+------------------------------------------------------------------+
//| CControlsDialog class                                            |
//| Objective: graphical panel for managing the application      |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CArrayObj         m_buttons;                     // button array
public:
                     CControlsDialog(void){};
                    ~CControlsDialog(void){};
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- add the button
   bool              AddButton(MyButton &button){return(m_buttons.Add(GetPointer(butto
protected:
   //--- create the buttons
   bool              CreateButtons(void);
  };
//+------------------------------------------------------------------+
//| Create the CControlsDialog object on the chart                   |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
   return(CreateButtons());
//---
  }
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Create and add buttons to the CControlsDialog panel             |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButtons(void)
  {
//--- calculate buttons coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2;
   int y2=y1+BUTTON_HEIGHT;
//--- add buttons objects together with pointers to functions
   AddButton(new MyButton("Open",Open));
   AddButton(new MyButton("Save",::Save));
   AddButton(new MyButton("Close",Close));
//--- create the buttons graphically
   for(int i=0;i<m_buttons.Total();i++)
     {
      MyButton *b=(MyButton*)m_buttons.At(i);
      x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
      x2=x1+BUTTON_WIDTH;
      if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
         PrintFormat("Failed to create button %s %d",b.Text(),i);
         return(false);
        }
      //--- add each button to the CControlsDialog container
      if(!Add(b))
         return(false);
     }
//--- succeed
   return(true);
  }
```

Now, we can develop the program using the CControlsDialog control panel having 3 buttons: Open, Save and Close. When clicking a button, the appropriate function in the form of the *TAction* pointer is called.

```
//--- declare the object on the global level to automatically create it when launching
CControlsDialog MyDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- now, create the object on the chart
   if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- launch the application
   MyDialog.Run();
//--- application successfully initialized
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy dialog
   MyDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
//--- call the handler from the parent class (here it is CAppDialog) for the chart eve
   MyDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

The launched application's appearance and button clicking results are provided on the screenshot.


EURUSD,H1
Panel_Buttons
Controls
Open
Save
Close
1.09415
1.09330
1.09245
1.09160
1.09075
1.08990
1.08905
1.08820
1.08735
1.08650
1.08565
30 Dec 2015
30 Dec 06:00
30 Dec 10:00
30 Dec 14:00
30 Dec 18:00
30 Dec 22:00
31 Dec 02:00
31 Dec 06:00
31 Dec 10:00
31 Dec 14:00
Toolbox
Trade
Exposure
History
News
Mailbox
Calendar
Company
Market
Alerts
Signals
Code Base

```
Message
Open function called (name=27031btOpen id=7)
Save function called (name=27031btSave id=8)
Close function called (name=27031btClose id=9)
```

The full source code of the program

```
//+------------------------------------------------------------------+
//|                                               Panel_Buttons.mq5 |
//|                        Copyright 2017, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+

#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The panel with several CButton buttons"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
```

```
//--- for buttons
#define BUTTON_WIDTH                            (100)     // size by X coordinate
#define BUTTON_HEIGHT                           (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                             (20)      // size by Y coordinate

//--- create the custom function type
typedef int(*TAction)(string,int);
//+------------------------------------------------------------------+
//|  Open the file                                                  |
//+------------------------------------------------------------------+
int Open(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(1);
  }
//+------------------------------------------------------------------+
//|  Save the file                                                  |
//+------------------------------------------------------------------+
int Save(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(2);
  }
//+------------------------------------------------------------------+
//|  Close the file                                                  |
//+------------------------------------------------------------------+
int Close(string name,int id)
  {
   PrintFormat("%s function called (name=%s id=%d)",__FUNCTION__,name,id);
   return(3);
  }
//+------------------------------------------------------------------+
//| Create the button class with the events processing function      |
//+------------------------------------------------------------------+
class MyButton: public CButton
  {
private:
   TAction           m_action;                    // chart events handler
public:
                     MyButton(void){}
                    ~MyButton(void){}
   //--- constructor specifying the button text and the pointer to the events handlin
                     MyButton(string text,TAction act)
     {
      Text(text);
      m_action=act;
     }
   //--- set the custom function called from the OnEvent() events handler
   void              SetAction(TAction act){m_action=act;}
   //--- standard chart events handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
     {
      if(m_action!=NULL && lparam==Id())
        {
         //--- call the custom handler
         m_action(sparam,(int)lparam);
         return(true);
        }
      else
      //--- return the result of calling the handler from the CButton parent class
         return(CButton::OnEvent(id,lparam,dparam,sparam));
     }
  };
//+------------------------------------------------------------------+
//| CControlsDialog class                                            |
//| Objective: graphical panel for managing the application       |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CArrayObj         m_buttons;                     // button array
public:
                     CControlsDialog(void){};
                    ~CControlsDialog(void){};
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- add the button
   bool              AddButton(MyButton &button){return(m_buttons.Add(GetPointer(butt
protected:
   //--- create the buttons
   bool              CreateButtons(void);
  };
//+------------------------------------------------------------------+
//| Create the CControlsDialog object on the chart                   |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
   return(CreateButtons());
//---
  }
//+------------------------------------------------------------------+
//| Create and add buttons to the CControlsDialog panel           |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButtons(void)
  {
```

```
//--- calculate buttons coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2;
   int y2=y1+BUTTON_HEIGHT;
//--- add buttons objects together with pointers to functions
   AddButton(new MyButton("Open",Open));
   AddButton(new MyButton("Save",::Save));
   AddButton(new MyButton("Close",Close));
//--- create the buttons graphically
   for(int i=0;i<m_buttons.Total();i++)
     {
      MyButton *b=(MyButton*)m_buttons.At(i);
      x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
      x2=x1+BUTTON_WIDTH;
      if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
         PrintFormat("Failed to create button %s %d",b.Text(),i);
         return(false);
        }
      //--- add each button to the CControlsDialog container
      if(!Add(b))
         return(false);
     }
//--- succeed
   return(true);
  }
//--- declare the object on the global level to automatically create it when launchin
CControlsDialog MyDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- now, create the object on the chart
   if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- launch the application
   MyDialog.Run();
//--- application successfully initialized
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy dialog
   MyDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
//--- call the handler from the parent class (here it is CAppDialog) for the chart eve
   MyDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

See also

Variables, Functions

Object Pointers

MQL5 enables the dynamic creation of complex type objects. This is done by using the ‘new’ operatorwhich returns a descriptor of the created object. The descriptor size is 8 bytes. Syntactically, object descriptors in MQL5 are similar to C++ pointers.

Example:

```
MyObject* hobject= new MyObject();
```

Unlike C++, the ‘hobject’ variable from the example above is not a pointer to memory, but is an object descriptor. Furthermore, in MQL5, all objects in function parameters must be passed by reference. The examples below show the passing of objects as function parameters:

```
class Foo
  {
public:
   string            m_name;
   int               m_id;
   static int        s_counter;
   //--- constructors and destructors
                     Foo(void){Setup("noname");};
                     Foo(string name){Setup(name);};
                    ~Foo(void){};
   //--- initialize the Foo object
   void              Setup(string name)
     {
      m_name=name;
      s_counter++;
      m_id=s_counter;
     }
  };
int Foo::s_counter=0;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the object as a variable, with automatic creation
   Foo foo1;
//--- variant of passing an object by reference
   PrintObject(foo1);

//--- declare a pointer to an object and create it using the 'new' operator
   Foo *foo2=new Foo("foo2");
//--- variant of passing a pointer to an object by reference
   PrintObject(foo2); // the pointer to the object is converted by the compiler automa

//--- declare an array of Foo objects
   Foo foo_objects[5];
//--- variant of passing an array of objects
   PrintObjectsArray(foo_objects); // a separate function for passing an array of obje

//--- declare an array of pointers to objects of type Foo
   Foo *foo_pointers[5];
   for(int i=0;i<5;i++)
      foo_pointers[i]=new Foo("foo_pointer");
//--- variant of passing an array of pointers
   PrintPointersArray(foo_pointers); // a separate function for passing an array of po

//--- before finishing, be sure to delete the objects created as pointers
   delete(foo2);
//--- remove the array of pointers
   int size=ArraySize(foo_pointers);
   for(int i=0;i<5;i++)
      delete(foo_pointers[i]);
//---
  }
//+------------------------------------------------------------------+
//|  Objects are always passed by reference                          |
//+------------------------------------------------------------------+
void PrintObject(Foo &object)
  {
   Print(__FUNCTION__,": ",object.m_id," Object name=",object.m_name);
  }
//+------------------------------------------------------------------+
//| Pass an array of objects                                         |
//+------------------------------------------------------------------+
void PrintObjectsArray(Foo &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
      PrintObject(objects[i]);
  }
//+------------------------------------------------------------------+
//| Pass an array of object pointers                                 |
//+------------------------------------------------------------------+
void PrintPointersArray(Foo* &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
      PrintObject(objects[i]);
  }
//+------------------------------------------------------------------+
```

Check the pointer before use

An attempt to access an invalid pointer causes the critical shutdown of the program. The CheckPointer function is utilized to check a pointer before use. The pointer can be invalid in the following cases:

- the pointer is equal to NULL;
- the object was destroyed using the delete operator.

This function can be used to validate of a pointer. A non-zero value indicates that data can be accessed at this pointer.

```
class CMyObject
  {
protected:
   double             m_value;
public:
                     CMyObject(void);
                     CMyObject(double value) {m_value=value;};
                    ~CMyObject(void){};
   //---
   double             Value(void) {return(m_value);}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create a non-initialized object
   CMyObject *pointer;
   if(CheckPointer(pointer)==POINTER_INVALID)
     Print("1. pointer is ", EnumToString(CheckPointer(pointer)));
   else
     Print("1. pointer.Value()=", pointer.Value());

//--- initialize the pointer
   pointer=new CMyObject(M_PI);
   if(CheckPointer(pointer)==POINTER_INVALID)
     Print("2. pointer is ", EnumToString(CheckPointer(pointer)));
   else
     Print("2. pointer.Value()=", pointer.Value());

//--- delete the object
   delete(pointer);
   if(CheckPointer(pointer)==POINTER_INVALID)
     Print("3. pointer is ", EnumToString(CheckPointer(pointer)));
   else
     Print("3. pointer.Value()=", pointer.Value());
  }
/*
    1. pointer is POINTER_INVALID
    2. pointer.Value()=3.141592653589793
    3. pointer is POINTER_INVALID
```

```
*/
```

To quickly validate the pointer, you can also use operator "!" (LNOT) which checks it via an implicit call of the CheckPointer function. This allows a more concise and clear code writing. Below are the checks from the previous example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- create a non-initialized object
  CMyObject *pointer;
  if(!pointer)
    Print("1. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("1. pointer.Value()=", pointer.Value());

//--- initialize the pointer
  pointer=new CMyObject(M_PI);
  if(!pointer)
    Print("2. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("2. pointer.Value()=", pointer.Value());

//--- delete the object
  delete(pointer);
  if(!pointer)
    Print("3. pointer is ", EnumToString(CheckPointer(pointer)));
  else
    Print("3. pointer.Value()=", pointer.Value());
 }
/*
   1. pointer is POINTER_INVALID
   2. pointer.Value()=3.141592653589793
   3. pointer is POINTER_INVALID
*/
```

Operator "==" is used for a quick check for NULL. For example: ptr==NULL or ptr!=NULL.

See also

Variables, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

References: Modifier & and Keyword this

Passing Parameters by Reference

In MQL5 parameters of simple types can be passed both by value and by reference, while parameters of compound types are always passed by reference. To inform the compiler that a parameter must be passed by reference, the ampersand character **&** is added before the parameter name.

Passing a parameter by reference means passing the address of the variable, that's why all changes in the parameter that is passed by reference will be immediately reflected in the source variable. Using parameter passing by reference, you can implement return of several results of a function at the same time. In order to prevent changing of a parameter passed by reference, use the const modifier.

Thus, if the input parameter of a function is an array, a structure or class object, symbol '&' is placed in the function header after the variable type and before its name.

Example

```
class CDemoClass
  {
private:
   double             m_array[];

public:
   void               setArray(double &array[]);
  };
//+------------------------------------------------------------------+
//| filling the array                                                |
//+------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
     ArrayResize(m_array,ArraySize(array));
     ArrayCopy(m_array, array);
     }
  }
```

In the above example class CDemoClass is declared, which contains the private member - array m_array[] of double type. Function setArray() is declared, to which array[] is passed by reference. If the function header doesn't contain the indication about passing by reference, i.e. doesn't contain the ampersand character, an error message will be generated at the attempt to compile such a code.

Despite the fact that the array is passed by reference, we can't assign one array to another. We need to perform the element-wise copying of contents of the source array to the recipient array. The presence of & in the function description is the obligatory condition for arrays and structures when passed as the function parameter.

Keyword this

A variable of class type (object) can be passed both by reference and by pointer. As well as reference, the pointer allows having access to an object. After the object pointer is declared, the new operator should be applied to it to create and initialize it.

The reserved word **this** is intended for obtaining the reference of the object to itself, which is available inside class or structure methods. **this** always references to the object, in the method of which it is used, and the expression GetPointer(this) gives the pointer of the object, whose member is the function, in which call of GetPointer() is performed. In MQL5 functions can't return objects, but they can return the object pointer.

Thus, if we need a function to return an object, we can return the pointer of this object in the form of GetPointer(this). Let's add function getDemoClass() that returns pointer of the object of this class, into the description of CDemoClass.

```
class CDemoClass
  {
private:
   double             m_array[];

public:
   void               setArray(double &array[]);
   CDemoClass        *getDemoClass();
  };
//+------------------------------------------------------------------+
//| filling the array                                                |
//+------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
      ArrayResize(m_array,ArraySize(array));
      ArrayCopy(m_array,array);
     }
  }
//+------------------------------------------------------------------+
//| returns its own pointer                                          |
//+------------------------------------------------------------------+
CDemoClass *CDemoClass::getDemoClass(void)
  {
   return(GetPointer(this));
  }
```

Structures don't have pointers, operators *new* and *delete* can't be applied to them, GetPointer(this) can't be used.

See also

Object Pointers, Creating and Deleting Objects, Visibility Scope and Lifetime of Variables

Operations and Expressions

Some characters and character sequences are of a special importance. These are so-called operation symbols, for example:

```
+ - * / %       Symbols of arithmetic operations
&& ||           Symbols of logical operations
= += *=         Characters assignment operators
```

Operation symbols are used in expressions and have sense when appropriate operands are given to them. Punctuation marks are emphasized, as well. These are parentheses, braces, comma, colon, and semicolon.

Operation symbols, punctuation marks, and spaces are used to separate language elements from each other.

This section contains the description of the following topics:

- Expressions
- Arithmetic Operations
- Assignment Operations
- Operations of Relation
- Boolean Operations
- Bitwise Operations
- Other Operations
- Priorities and Operations Order

Expressions

An expression consists of one or more operands and operation symbols. An expression can be written in several lines.

Examples:

```
a++; b = 10;          // several expressions are located in one line
//---  one expression is divided into several lines
x = (y * z) /
    (w + 2) + 127;
```

An expression that ends with a semicolon (;) is an operator.

See also

Precedence Rules

Arithmetic Operations

Arithmetic operations include additive and multiplicative operations:

```
Sum of variables                          i = j + 2;
Difference of variables                   i = j - 3;
Changing the sign                         x = - x;
Product of variables                      z = 3 * x;
Division quotient                         i = j / 5;
Remainder of division                     minutes = time % 60;
Adding 1 to the variable value            i++;
Adding 1 to the variable value            ++i;
Subtracting 1 from the variable value     k--;
Subtracting 1 from the variable value     --k;
```

Increment and decrement operations are applied only to variables, they can't be applied to constants. The prefix increment (++i) and decrement (--k) are applied to the variable right before this variable is used in an expression.

Post-increment (i++) and post-decrement (k--) are applied to the variable right after this variable is used in an expression.

Important Notice

```
int i=5;
int k = i++ + ++i;
```

Computational problems may occur while moving the above expression from one programming environment to another one (for example, from Borland C++ to MQL5). In general, the order of computations depends on the compiler implementation. In practice, there are two ways to implement the post-decrement (post-increment):

1. The post-decrement (post-increment) is applied to the variable after calculating the whole expression.
2. The post-decrement (post-increment) is applied to the variable immediately at the operation.

Currently the first way of post-decrement (post-increment) calculation is implemented in MQL5. But even knowing this peculiarity, it is not recommended to experiment with its use.

Examples:

```
int a=3;
a++;            // valid expression
int b=(a++)*3;  // invalid expression
```

See also

Precedence Rules

Assignment Operations

The value of the expression that includes the given operation is the value of the left operand after assignment:

Assigning the value of x to the y variable	y = x;

The following operations unite arithmetic or bitwise operations with operation of assignment:

Adding x to the y variable	y += x;
Subtracting x from the y variable	y -= x;
Multiplying the y variable by x	y *= x;
Dividing the y variable by x	y /= x;
Reminder of division of the y variable by x	y %= x;
Shift of the binary representation of y to the right by x bits	y >>= x;
Shift of the binary representation of y to the left by x bits	y <<= x;
AND bitwise operation of binary representations of y and x	y &= x;
OR bitwise operation of binary representations of y and x	y |= x;
Excluding OR bitwise operation of binary representations of y and x	y ^= x;

Bitwise operations can be applied to integers only. When performing the operation of the logical shift of the y representation to the right/left by x bits, the 5 smallest binary digits of the x value are used, the highest ones are dropped, i.e. the shift is made to 0-31 bits.

By %= operation (y value by module of x), the result sign is equal to the sign of divided number.

The assignment operator can be used several times in an expression . In this case the processing of the expression is performed from left to right:

```
 y=x=3;
```

First, the variable x will be assigned the value 3, then the y variable will be assigned the value of x, i.e. also 3.

See also

Precedence Rules

Operations of Relation

Boolean FALSE is represented with an integer zero value, while the boolean TRUE is represented by any non-zero value.

The value of expressions containing operations of relation or logical operations is FALSE (0) or TRUE (1).

```
True if a is equal to b                     a == b;
True if a is not equal to b                 a != b;
True if a is less than b                    a < b;
True if a is greater than b                 a > b;
True if a is less than or equal to b        a <= b;
True if a is greater than or equal to b     a >= b;
```

The equality of two real numbers can't be compared. In most cases, two seemingly identical numbers can be unequal because of different values in the 15th decimal place. In order to correctly compare two real numbers, compare the normalized difference of these numbers with zero.

Example:

```
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0) return(true);
   else return(false);
  }
void OnStart()
  {
   double first=0.3;
   double second=3.0;
   double third=second-2.7;
   if(first!=third)
     {
      if(CompareDoubles(first,third))
         printf("%.16f and %.16f are equal",first,third);
     }
  }
// Result: 0.3000000000000000  0.2999999999999998   are equal
```

See also

Precedence Rules

Boolean Operations

Logical Negation NOT (!)

Operand of the logical negation (!) must be of arithmetic type. The result is TRUE (1), if the operand value is FALSE (0); and it is equal to FALSE (0), if the operand differs from FALSE (0).

```
if(!a) Print("not 'a'");
```

Logical Operation OR (||)

Logical OR operation (||) of x and y values. The expression value is TRUE (1), if x or y value is true (not null). Otherwise - FALSE (0).

```
if(x<0 || x>=max_bars) Print("out of range");
```

Logical Operation AND (&&)

Logical operation AND (&&) of x and y values. The expression value is TRUE (1), if the values of x and y are true (not null). Otherwise - FALSE (0).

Brief Estimate of Boolean Operations

The scheme of the so called "brief estimate" is applied to boolean operations, i.e. the calculation of the expression is terminated when the result of the expression can be precisely estimated.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- the first example of the brief estimate
   if(func_false() && func_true())
     {
      Print("Operation &&: You will never see this expression");
     }
   else
     {
      Print("Operation &&: Result of the first expression is false, so the second wasn
     }
//--- the second example of the brief estimate
   if(!func_false() || !func_true())
     {
      Print("Operation ||: Result of the first expression is true, so the second wasn'
     }
   else
     {
      Print("Operation ||: You will never see this expression");
     }
  }
//+------------------------------------------------------------------+
//| the function always returns false                                |
//+------------------------------------------------------------------+
bool func_false()
  {
   Print("Function func_false()");
   return(false);
  }
//+------------------------------------------------------------------+
//| the function always returns true                                 |
//+------------------------------------------------------------------+
bool func_true()
  {
   Print("Function func_true()");
   return(true);
  }
```

See also

Precedence Rules

Bitwise Operations

Complement to One

Complement of the variable value up to one. The value of the expression contains 1 in all digits where the variable value contains 0, and 0 in all digits where the variable contains 1.

```
b = ~n;
```

Example:

```
   char a='a',b;
   b=~a;
   Print("a = ",a, "  b = ",b);
// The result will be:
// a = 97   b = -98
```

Right Shift

The binary representation of x is shifted to the right by y digits. If the value to shift is of the unsigned type, the logical right shift is made, i.e. the freed left-side bits will be filled with zeroes.

If the value to shift is of a sign type, the arithmetic right shift is made, i.e. the freed left-side digits will be filled with the value of a sign bit (if the number is positive, the value of the sign bit is 0; if the number is negative, the value of the sign bit is 1).

```
x = x >> y;
```

Example:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- shift to the right
   b=a>>1;
   Print("After:   a = ",a, "  b = ",b);
// The result will be:
// Before:  a = 97   b = 98
// After:   a = 97   b = 48
```

Left Shift

The binary representation of x is shifted to the left by y digits, the freed right-side digits are filled with zeros.

```
x = x << y;
```

Example:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- shift to the left
   b=a<<1;
   Print("After:   a = ",a, "  b = ",b);
```

```
// The result will be:
// Before:  a = 97   b = 98
// After:   a = 97   b = -62
```

It is not recommended to shift by the number of bits larger or equal to the length of the variable shifted, because the result of such an operation is undefined.

Bitwise AND Operation

The bitwise AND operation of binary-coded x and y representations. The value of the expression contains a 1 (TRUE) in all digits where both x and y contain non-zero, and it contains 0 (FALSE) in all other digits.

```
b = ((x & y) != 0);
```

Example:

```
   char a='a',b='b';
//--- AND operation
   char c=a&b;
   Print("a = ",a,"  b = ",b);
   Print("a & b = ",c);
// The result will be:
// a = 97   b = 98
// a & b = 96
```

Bitwise OR Operation

The bitwise OR operation of binary representations of x and y. The value of the expression contains 1 in all digits where x or y does not contain 0, and it contains 0 in all other digits.

```
b = x | y;
```

Example:

```
   char a='a',b='b';
//--- OR operation
   char c=a|b;
   Print("a = ",a,"  b = ",b);
   Print("a | b = ",c);
// The result will be:
// a = 97   b = 98
// a | b = 99
```

Bitwise Exclusive Operation OR

The bitwise exclusive OR (eXclusive OR) operation of binary representations of x and y. The value of the expression contains a 1 in all digits where x and y have different binary values, and it contains 0 in all other digits.

```
b = x ^ y;
```

Example:

```
   char a='a', b='b';
//--- Excluding OR operation
   char c=a^b;
   Print("a = ",a,"  b = ",b);
   Print("a ^ b = ",c);
// The result will be:
// a = 97   b = 98
// a ^ b = 3
```

Bitwise operations are performed with integers only.

See also

Precedence Rules

Other operations

Indexing ([])

When addressing the i-th element of the array, the expression value is the value of a variable with the serial number i.

Example:

```
array[i] = 3; // Assign the value of 3 to i-th element of the array.
```

Only an integer can be index of an array. Four-dimensional and below arrays are allowed. Each dimension is indexed from 0 to **dimension size**-1. In particular case, for a one-dimensional array consisting of 50 elements, the reference to the first element will look like array [0], that to the last element will be array [49].

When addressing beyond the array, the executing subsystem will generate a critical error, and the program will be stopped.

Calling Function with x1, x2 ,..., xn Arguments

Each argument can represent a constant, variable, or expression of the corresponding type. The arguments passed are separated by commas and must be inside of parentheses, the opening parenthesis must follow the name of the called function.

The expression value is the value returned by the function. If the return value is of void type, such function call cannot be placed to the right in the assignment operation. Note that the expressions x1,..., xn are executed exactly in this order.

Example:

```
   int length=1000000;
   string a="a",b="b",c;
//---Other Operations
   int start=GetTickCount(),stop;
   long i;
   for(i=0;i<length;i++)
     {
      c=a+b;
     }
   stop=GetTickCount();
   Print("time for 'c = a + b' = ",(stop-start)," milliseconds, i = ",i);
```

Comma Operation (,)

Expressions separated by commas are executed from left to right. All side effects of the left expression calculation can appear before the right expression is calculated. The result type and value coincide with those of the right expression. The list of parameters to be passed (see above) can be considered as an example.

Example:

```
for(i=0,j=99; i<100; i++,j--) Print(array[i][j]);
```

Dot Operator (.)

For the direct access to the public members of structures and classes the dot operation is used. Syntax:

```
Variable_name_of_structure_type.Member_name
```

Example:

```
   struct SessionTime
     {
      string sessionName;
      int    startHour;
      int    startMinutes;
      int    endHour;
      int    endMinutes;
     } st;
   st.sessionName="Asian";
   st.startHour=0;
   st.startMinutes=0;
   st.endHour=9;
   st.endMinutes=0;
```

Scope Resolution Operation (::)

Each function in a mql5 program has its own execution scope. For example, the Print() system function is performed in a global scope. Imported functions are called in the scope of the corresponding import. Method functions of classes have the scope of the corresponding class. The syntax of the scope resolution operation is as follows:

```
[Scope_name]::Function_name(parameters)
```

If there is no scope name, this is the explicit direction to use the global scope. If there is no scope resolution operation, the function is sought in the nearest scope. If there is no function in the local scope, the search is conducted in the global scope.

The scope resolution operation is also used to define function-class member.

```
type Class_name::Function_name(parameters_description)
   {
// function body
   }
```

Use of several functions of the same name from different execution contexts in a program may cause ambiguity. The priority order of function calls without explicit scope specification is the following:

1. Class methods. If no function with the specified name is set in the class, move to the next level.
2. MQL5 functions. If the language does not have such a function, move to the next level.
3. User defined global functions. If no function with the specified name is found, move to the next level.

4. Imported functions. If no function with the specified name is found, the compiler returns an error.

To avoid the ambiguity of function calls, always explicitly specify the function scope using the scope resolution operation.

Example:

```
#property script_show_inputs
#import "kernel32.dll"
   int GetLastError(void);
#import

class CCheckContext
  {
   int         m_id;
public:
               CCheckContext() { m_id=1234; }
protected:
   int         GetLastError() { return(m_id); }
  };
class CCheckContext2 : public CCheckContext
  {
   int         m_id2;
public:
               CCheckContext2() { m_id2=5678; }
   void        Print();
protected:
   int         GetLastError() { return(m_id2); }
  };
void CCheckContext2::Print()
  {
   ::Print("Terminal GetLastError",::GetLastError());
   ::Print("kernel32 GetLastError",kernel32::GetLastError());
   ::Print("parent GetLastError",CCheckContext::GetLastError());
   ::Print("our GetLastError",GetLastError());
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CCheckContext2 test;
   test.Print();
  }
//+------------------------------------------------------------------+
```

Operation of Obtaining Data Type Size or Size of Any Data Type Object (sizeof)

Using the sizeof operation, the memory size corresponding to an identifier or type can be defined. The sizeof operation is of the following format:

Example:

```
sizeof(expression)
```

Any identifier, or type name enclosed in brackets can be used as an expression. Note that the void type name can't be used, and the identifier cannot belong to the field of bits, or be a function name.

If the expression is the name of a static array (i.e. the first dimension is given), then the result is the size of the whole array (i.e. the product of the number of elements and the length of the type). If the expression is the name of a dynamic array (the first dimension is not specified), the result will be the size of the object of the dynamic array.

When sizeof is applied to the name of structure or class type, or to the identifier of the structure or class type, the result is the actual size of the structure or class.

Example:

```
   struct myStruct
     {
      char   h;
      int    b;
      double f;
     } str;
   Print("sizeof(str) = ",sizeof(str));
   Print("sizeof(myStruct) = ",sizeof(myStruct));
```

The size is calculated at the compilation stage.

See also

Precedence Rules

Precedence Rules

Each group of operations in the table has the same priority. The higher the priority of operations is, the higher it is position of the group in the table. The precedence rules determine the grouping of operations and operands.

Attention: Precedence of operations in the MQL5 language corresponds to the priority adopted in C++, and differs from the priority given in the MQL4 language.

Operation	Desciption	Execution Order
() [] .	Function Call Referencing to an array element Referencing to a structure element	From left to right
! ~ - ++ -- (type) sizeof	Logical negation Bitwise negation (complement) Sign changing Increment by one Decrement by one Typecasting Determining size in bytes	Right to left
* / %	Multiplication Division Module division	From left to right
+ -	Addition Subtraction	From left to right
<< >>	Left shift Right shift	From left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	From left to right
== !=	Equal Not equal	From left to right
&	Bitwise AND operation	From left to right
^	Bitwise exclusive OR	From left to right
|	Bitwise OR operation	From left to right
&&	Logical AND operation	From left to right
||	Logical OR operation	From left to right
?:	Conditional Operator	Right to left
= *= /=	Assignment Multiplication with assignment Division with assignment	Right to left

Operation	Desciption	Execution Order
%= += -= <<= >>= &= ^= |=	Module with assignment Addition with assignment Subtraction with assignment Left shift with assignment Right shift with assignment Bitwise AND with assignment Exclusive OR with assignment Bitwise OR with assignment	
,	Comma	From left to right

To change the operation execution order, parenthesis that are of higher priority are used.

Operators

Language operators describe some algorithmic operations that must be executed to accomplish a task. The program body is a sequence of such operators. Operators following one by one are separated by semicolons.

Operator	Description
Compound operator {}	One or more operators of any type, enclosed in curly braces {}
Expression operator (;)	Any expression that ends with a semicolon (;)
return operator	Terminates the current function and returns control to the calling program
if-else conditional operator	Is used when it's necessary to make a choice
?: conditional operator	A simple analog of the if-else conditional operator
switch selection operator	Passes control to the operator, which corresponds to the expression value
while loop operator	Performs an operator until the expression checked becomes false. The expression is checked before each iteration
for loop operator	Performs an operator until the expression checked becomes false. The expression is checked before each iteration
do-while loop operator	Performs an operator until the expression checked becomes false. The end condition is checked, after each loop. The loop body is always executed at least once.
break operator	Terminates the execution of the nearest attached external operator switch, while, do-while or for
continue operator	Passes control to the beginning of the nearest external loop operator while, do-while or for
@ operator	Implements matrix multiplication according to the rules of linear algebra. It allows multiplying matrices and vectors, as well as performing scalar multiplication of vectors.
new operator	Creates an object of the appropriate size and returns a descriptor of the created object.
delete operator	Deletes the object created by the new operator

One operator can occupy one or more lines. Two or more operators can be located in the same line. Operators that control over the execution order (if, if-else, switch, while and for), can be nested into each other.

Example:

```
if(Month() == 12)
  if(Day() == 31) Print("Happy New Year!");
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Compound Operator

A compound operator (a block) consists of one or more operators of any type, enclosed in braces {}. The closing brace must not be followed by a semicolon (;).

Example:

```
if(x==0)
  {
   Print("invalid position x = ",x);
   return;
  }
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Expression Operator

Any expression followed by a semicolon (;) is the operator. Here are some examples of expression operators.

Assignment Operator

Identifier = expression;

```
x=3;
y=x=3;
bool equal=(x==y);
```

Assignment operator can be used many times in an expression. In this case, the expression is processed from right to left.

Function Calling Operator

Function_name (argument1,..., argumentN);

```
FileClose(file);
```

Empty Operator

Consists only of a semicolon (;) and is used to denote an empty body of a control operator.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Return Operator

The return operator terminates the current function execution and returns control to the calling program. The expression calculation result is returned to the calling function. The expression can contain an assignment operator.

Example:

```
int CalcSum(int x, int y)
  {
   return(x+y);
  }
```

In functions with the void return type, the return operator without expression must be used:

```
void SomeFunction()
  {
   Print("Hello!");
   return;    // this operator can be removed
  }
```

The right brace of the function means implicit execution of the return operator without expression.

What can be returned: simple types, simple structures, object pointers. With the *return* operator you can't return any arrays, class objects, variables of compound structure type.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

If-Else Conditional Operator

The IF - ELSE operator is used when a choice must be made. Formally, the syntax is as follows:

```
if (expression)
     operator1
else
     operator2
```

If the expression is true, operator1 is executed and control is given to the operator that follows operator2 (operator2 is not executed). If the expression is false, operator2 is executed.

The else part of the if operator can be omitted. Thus, a divergence may appear in nested if operators with omitted else part. In this case, else addresses to the nearest previous if operator in the same block that has no else part.

Examples:

```
//--- The else part refers to the second if operator:
if(x>1)
   if(y==2) z=5;
else     z=6;
//--- The else part refers to the first if operator:
if(x>l)
  {
   if(y==2) z=5;
  }
else        z=6;
//--- Nested operators
if(x=='a')
  {
   y=1;
  }
else if(x=='b')
  {
   y=2;
   z=3;
  }
else if(x=='c')
  {
   y=4;
  }
else Print("ERROR");
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Ternary Operator ?:

The general form of the ternary operator is as follows:

```
expression1 ? expression2 : expression3
```

For the first operand - "expression1" - any expression that results in a bool type value can be used. If the result is true, then the operator set by the second operand, i.e. "expression2" is executed.

If the first operand is false, the third operand - "expression3" is performed. The second and third operands, i.e. "expression2" and "expression3" should return values of one type and should not be of void type. The result of the conditional operator execution is the result of expression2 or result of the expression3, depending on the result of expression1.

```
//--- normalize difference between open and close prices for a day range
double true_range = (High==Low)?0:(Close-Open)/(High-Low);
```

This entry is equivalent to the following:

```
   double true_range;
   if(High==Low)true_range=0;               // if High and Low are equal
   else true_range=(Close-Open)/(High-Low); // if the range is not null
```

Operator Use Restrictions

Based on the value of "expression1", the operator must return one of the two values - either "expression2" or "expression3". There are several limitations to these expressions:

1. Do not mix user-defined type with simple type or enumeration. NULL can be used for the pointer.
2. If types of values are simple, the operator will be of the maximum type (see Type casting).
3. If one of the values is an enumeration and the second one is of a numeric type, the enumeration is replaced by int and the second rule is applied.
4. If both values are enumerations, their types must be identical, and the operator will be of type enumeration.

Restrictions for the user-defined types (classes or structures):

a) Types must be identical or one should be derived from the other one.

b) If types are not identical (inheritance), then the child is implicitly cast to the parent, i.e. the operator will be of the parent type.

c) Do not mix object and the pointer – both expressions must be either objects or pointers. NULL can be used for the pointer.

Note

Be careful when using the conditional operator as an argument of an overloaded function, because the type of the result of a conditional operator is defined at the time of program compilation. And this type is determined as the larger of the types "expression2" and "expression3".

Example:

```
void func(double d) { Print("double argument: ",d); }
void func(string s) { Print("string argument: ",s); }

bool   Expression1=true;
double Expression2=M_PI;
string Expression3="3.1415926";

void OnStart()
  {
   func(Expression2);
   func(Expression3);

   func(Expression1?Expression2:Expression3);   // warning on implicit casting to str
   func(!Expression1?Expression2:Expression3);  // warning on implicit casting to str
  }

//   Result:
//   double argument: 3.141592653589793
//   string argument: 3.1415926
//   string argument: 3.141592653589793
//   string argument: 3.1415926
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Switch Operator

Compares the expression value with constants in all the *case* variants and passes control to the operator that corresponds to the expression value. Each variant of *case* can be marked with an integer constant, a literal constant or a constant expression. The constant expression can't contain variables or function calls. Expression of the *switch* operator must be of integer type – int or uint.

```
switch(expression)
  {
   case constant: operators
   case constant: operators
      ...
   default: operators
  }
```

Operators marked by the *default* label are executed if none of the constants in *case* operators is equal to the expression value. The *default* variant should not be necessarily declared and should not be necessarily the last one. If none of the constants corresponds to the expression value and the *default* variant is not available, no actions are executed.

The *case* keyword with a constant are just labels, and if operators are executed for some *case* variant, the program will further execute the operators of all subsequent variants until the *break* operator occurs. It allows to bind a sequence of operators with several variants.

A constant expression is calculated during compilation. No two constants in one *switch* operator can have the same value.

Examples:

```
//--- First example
switch(x)
  {
   case 'A':
      Print("CASE A");
      break;
   case 'B':
   case 'C':
      Print("CASE B or C");
      break;
   default:
      Print("NOT A, B or C");
      break;
  }

//---  Second example
   string res="";
   int i=0;
   switch(i)
     {
      case 1:
         res=i;break;
      default:
         res="default";break;
      case 2:
         res=i;break;
      case 3:
         res=i;break;
     }
   Print(res);
/*
   Result
   default
*/
```

See also

[Initialization of Variables](), [Visibility Scope and Lifetime of Variables](), [Creating and Deleting Objects]()

While Loop Operator

The while operator consists of a checked expression and the operator, which must be fulfilled:

```
while(expression)
  operator;
```

If the expression is true, the operator is executed until the expression becomes false. If the expression is false, the control is passed to the next operator. The expression value is defined before the operator is executed. Therefore, if the expression is false from the very beginning, the operator will not be executed at all.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Example:

```
while(k<n && !IsStopped())
  {
   y=y*x;
   k++;
  }
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

For Loop Operator

The for operator consists of three expressions and an executable operator:

```
for(expression1; expression2; expression3)
   operator;
```

Expression1 describes the loop initialization. Expression2 checks the conditions of the loop termination. If it is true, the loop body for is executed. The loop repeats expression2 until it becomes false. If it is false, the loop is terminated, and control is given to the next operator. Expression3 is calculated after each iteration.

The for operator is equivalent to the following succession of operators:

```
expression1;
while(expression2)
  {
   operator;
   expression3;
  };
```

Any of the three or all three expressions can be absent in the for operator, but the semicolons (;) that separate them must not be omitted. If expression2 is omitted, it is considered constantly true. The for(;;) operator is a continuous loop, equivalent to the while(1) operator. Each expression 1 or 3 can consist of several expressions combined by a comma operator ','.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Examples:

```
for(x=1;x<=7000; x++)
  {
   if(IsStopped())
      break;
   Print(MathPower(x,2));
  }
//--- Another example
for(;!IsStopped();)
  {
   Print(MathPower(x,2));
   x++;
   if(x>10) break;
  }
//--- Third example
for(i=0,j=n-l;i<n && !IsStopped();i++,j--) a[i]=a[j];
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Loop Operator do while

The for and while loops check the termination at the beginning, not at the end of a loop. The third loop operator do - while checks the condition of termination at the end, after each loop iteration. The loop body is always executed at least once.

```
do
   operator;
while(expression);
```

First the operator is executed, then the expression is calculated. If it is true, then the operator is executed again, and so on. If the expression becomes false, the loop terminates.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Example:

```
//--- Calculate the Fibonacci series
   int counterFibonacci=15;
   int i=0,first=0,second=1;
   int currentFibonacciNumber;
   do
     {
      currentFibonacciNumber=first+second;
      Print("i = ",i,"  currentFibonacciNumber = ",currentFibonacciNumber);
      first=second;
      second=currentFibonacciNumber;
      i++; // without this operator an infinite loop will appear!
     }
   while(i<counterFibonacci && !IsStopped());
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Break Operator

The break operator terminates the execution of the nearest nested outward switch, while, do-while or for operator. The control is passed to the operator that follows the terminated one. One of the purposes of this operator is to finish the looping execution when a certain value is assigned to a variable.

Example:

```
//--- searching for the first zero element
for(i=0;i<array_size;i++)
  if(array[i]==0)
    break;
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Continue Operator

The continue operator passes control to the beginning of the nearest outward loop while, do-while or for operator, the next iteration being called. The purpose of this operator is opposite to that of break operator.

Example:

```
//--- Sum of all nonzero elements
int func(int array[])
  {
   int array_size=ArraySize(array);
   int sum=0;
   for(int i=0;i<array_size; i++)
     {
      if(a[i]==0) continue;
      sum+=a[i];
     }
   return(sum);
  }
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Matrix product operator @

The @ MQL5 operator implements matrix multiplication according to the rules of linear algebra. It allows multiplying matrices and vectors, as well as performing scalar multiplication of vectors.

Supported element types:

- float
- double
- complex<float>
- complex<double>

Important: The element types in the left and right operands must match.

Examples of use

1. Matrix multiplication (matrix × matrix)

```
matrix A(2, 3);
matrix B(3, 2);
matrix C = A @ B; // Result: Matrix C of size [2,2]
```

2. Matrix multiplication (matrix × vector)

```
matrix M(2, 3);
vector V(3);
vector R = M @ V; // Result: Vector R of 2 elements
```

3. Matrix multiplication (vector x matrix)

```
matrix M(2, 3);
vector V(1, 2);
vector R = V @ M; // Result: Vector R of 3 elements
```

4. Scalar multiplication (vector × vector)

```
vector V1(1, 3), V2(1, 3);
double r = V1 @ V2; // Result: Scalar
```

Note

The dimensions must comply with the rules of multiplication: the number of columns in the first operand = the number of rows in the second.

In case of a dimension error, an exception will be triggered at runtime.

Vectors on the left are considered horizontal (1×n).

Vectors on the right are considered vertical (n×1).

The priority of the @ operation corresponds to the priority of multiplication, i.e. for an entry of D=C+A@B, first matrix multiplication T=A@B is performed, followed by the element-wise addition D=C+T.

See also

MatMul, GeMM, Kron, Dot,

Object Create Operator new

The new operator automatically creates an object of a corresponding size, calls the object constructor and returns a descriptor of created object. In case of failure, the operator returns a null descriptor that can be compared with the NULL constant.

The new operator can be applied only to class objects. It can't be applied to structures.

The operator shall not be used to create arrays of objects. To do this, use the ArrayResize() function.

Example:

```
//+------------------------------------------------------------------+
//| Figure creation                                                  |
//+------------------------------------------------------------------+
void CTetrisField::NewShape()
  {
   m_ypos=HORZ_BORDER;
//--- randomly create one of the 7 possible shapes
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- draw
   if(m_shape!=NULL)
     {
      //--- pre-settings
      m_shape.SetRightBorder(WIDTH_IN_PIXELS+VERT_BORDER);
      m_shape.SetYPos(m_ypos);
      m_shape.SetXPos(VERT_BORDER+SHAPE_SIZE*8);
      //--- draw
      m_shape.Draw();
     }
//---
  }
```

It should be noted that object descriptor is not a pointer to memory address.

An object created with the new operator must be explicitly removed using the delete operator.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Object Delete Operator delete

The delete operator deletes an object created by the new operator, calls the corresponding class destructor and frees up memory occupied by the object. A real descriptor of an existing object is used as an operand. After the delete operation is executed, the object descriptor becomes invalid.

Example:

```
       //--- delete figure
       delete m_shape;
       m_shape=NULL;
       //--- create a new figure
       NewShape();
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Functions

Every task can be divided into subtasks, each of which can either be directly represented in the form of a code, or divided into smaller sub-tasks. This method is called *stepwise refinement*. Functions are used for writing the code of sub-tasks to be solved. The code that describes what a function does is called *function definition:*

```
function_header
  {
   instructions
  }
```

All that is before the first brace is the *header* of the function definition, and what is between braces is the *body* of the function definition. The function header includes a description of the return value type, name (identifier) and formal parameters. The number of parameters passed to the function is limited and cannot exceed 64.

The function can be called from other parts of the program as many times as necessary. In fact, the return type, function identifier and parameter types constitute the *function prototype*.

Function prototype is the function declaration, but not its definition. Due to the explicit declaration of the return type and a list of argument types, the strict type checking and implicit typecasting are possible during function calls. Very often function declarations are used in classes to improve the code readability.

The function definition must exactly match its declaration. Each declared function must be defined.

Example:

```
double                        // return value type
linfunc (double a, double b)  // function name and parameter list
  {
                              // composite operator
   return (a + b);            // return value
  }
```

The return operator can return the value of an expression located in this operator. If necessary, the expression value is converted to the function result type. What can be returned: simple types, simple structures, object pointers. With the *return* operator you can't return any arrays, class objects, variables of compound structure type.

A function that returns no value should be described as that of void type.

Example:

```
void errmesg(string s)
  {
   Print("error: "+s);
  }
```

Parameters passed to the function can have default values, which are defined by constants of that type.

Example:

```
int somefunc(double a,
             double d=0.0001,
             int n=5,
             bool b=true,
             string s="passed string")
  {
   Print("Required parameter a = ",a);
   Print("Pass the following parameters: d = ",d," n = ",n," b = ",b," s = ",s);
   return(0);
  }
```

If any of parameters has a default value, all subsequent parameters must also have default values.

Example of incorrect declaration:

```
int somefunc(double a,
             double d=0.0001,    // default value 0.0001 declared
             int n,              // default value is not specified !
             bool b,             // default value is not specified !
             string s="passed string")
  {
  }
```

See also

[Overload](), [Virtual Functions](), [Polymorphism]()

Function Call

If a name that has not been described before, appears in the expression and is followed by the left parenthesis, it is contextually considered as the name of a function.

```
function_name (x1, x2,..., xn)
```

Arguments (formal parameters) are passed by value, i.e. each expression x1,..., xn is calculated, and the value is passed to the function. The order of expressions calculation and the order of values loading are not guaranteed. During the execution, the system checks the number and type of arguments passed to the function. Such way of addressing to the function is called a value call.

Function call is an expression, the value of which is the value returned by the function. The function type described above must correspond with the type of the return value. The function can be declared or described in any part of the program on the global scope, i.e., outside other functions. The function cannot be declared or described inside of another function.

Examples:

```
int start()
  {
   double some_array[4]={0.3, 1.4, 2.5, 3.6};
   double a=linfunc(some_array, 10.5, 8);
   //...
  }
double linfunc(double x[], double a, double b)
  {
   return (a*x[0] + b);
  }
```

At calling of a function with default parameters, the list of parameters to be passed can be limited, but not before the first default parameter.

Examples:

```
void somefunc(double init,
              double sec=0.0001, //set default values
              int level=10);
//...
somefunc();                      // Wrong call. The first parameter must be presented.
somefunc(3.14);                  // Correct call
somefunc(3.14,0.0002);           // Correct call
somefunc(3.14,0.0002,10);        // Correct call
```

When calling a function, one may not skip parameters, even those having default values:

```
somefunc(3.14, , 10);           // Wrong call -> the second parameter was skipped.
```

See also

Overload, Virtual Functions, Polymorphism

Passing Parameters

There are two methods, by which the machine language can pass arguments to a subprogram (function). The first method is to send a parameter by value. This method copies the argument value into a formal function parameter. Therefore, any changes in this parameter within the function have no influence on the corresponding call argument.

```
//+------------------------------------------------------------------+
//| Passing parameters by value                                      |
//+------------------------------------------------------------------+
double FirstMethod(int i,int j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=FirstMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//--- Result of script execution
//  a and b before call: 14 8
//  a and b after call: 14 8
```

The second method is to pass by reference. In this case, reference to a parameter (not its value) is passed to a function parameter. Inside the function, it is used to refer to the actual parameter specified in the call. This means that the parameter changes will affect the argument used to call the function.

```
//+------------------------------------------------------------------+
//| Passing parameters by reference                                  |
//+------------------------------------------------------------------+
double SecondMethod(int &i,int &j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=SecondMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//+------------------------------------------------------------------+
//--- result of script execution
//  a and b before call: 14 8
//  a and b after call: 28 4
```

MQL5 uses both methods, with one exception: arrays, structure type variables and class objects are always passed by reference. In order to avoid changes in actual parameters (arguments passed at function call) use the access specifier const. When trying to change the contents of a variable declared with the *const* specifier, the compiler will generate an error.

Note

It should be noted that parameters are passed to a function in reversed order, i.e., first the last parameter is calculated and passed, and then the last but one, etc. The last calculated and passed parameter is the one that stands first after opening parenthesis.

Example:

```
void OnStart()
  {
//---
   int a[]={0,1,2};
   int i=0;

   func(a[i],a[i++],"First call (i = "+string(i)+")");
   func(a[i++],a[i],"Second call (i = "+string(i)+")");
// Result:
// First call (i = 0) : par1 = 1     par2 = 0
// Second call (i = 1) : par1 = 1     par2 = 1

  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void func(int par1,int par2,string comment)
  {
   Print(comment,": par1 = ",par1,"    par2 = ",par2);
  }
```

In first call (see example above) the ***i*** variable is first used in strings concatenation:

```
  "First call (i = "+string(i)+")"
```

Here its value doesn't change. Then the ***i*** variable is used in calculation of the ***a[i++]*** array element, i.e. when array element with index i is accessed, the ***i*** variable is incremented. And only after that the first parameter with changed value of ***i*** variable is calculated.

In the second call the same value of i (calculated on the first phase of function calling) is used when calculating all three parameters. Only after the first parameters is calculated the ***i*** variable is changed again.

See also

Visibility Scope and Lifetime of Variables, Overload, Virtual Functions, Polymorphism

Function Overloading

Usually the function name tends to reflect its main purpose. As a rule, readable programs contain various well selected identifiers. Sometimes different functions are used for the same purposes. Let's consider, for example, a function that calculates the average value of an array of double precision numbers and the same function, but operating with an array of integers. Both are convenient to be called AverageFromArray:

```
//+------------------------------------------------------------------+
//| The calculation of average for an array of double type           |
//+------------------------------------------------------------------+
double AverageFromArray(const double & array[],int size)
  {
   if(size<=0) return 0.0;
   double sum=0.0;
   double aver;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];    // Summation for the double
     }
   aver=sum/size;       // Just divide the sum by the number
//---
   Print("Calculation of the average for an array of double type");
   return aver;
  }
//+------------------------------------------------------------------+
//| The calculation of average for an array of int type              |
//+------------------------------------------------------------------+
double AverageFromArray(const int & array[],int size)
  {
   if(size<=0) return 0.0;
   double aver=0.0;
   int sum=0;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];     // Summation for the int
     }
   aver=(double)sum/size;// Give the amount of type double, and divide
//---
   Print("Calculation of the average for an array of int type");
   return aver;
  }
```

Each function contains the message output via the Print() function;

```
   Print("Calculation of the average for an array of int type");
```

The compiler selects a necessary function in accordance with the types of arguments and their quantity. The rule, according to which the choice is made, is called the *signature matching algorithm*. A signature is a list of types used in the function declaration.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int    a[5]={1,2,3,4,5};
   double b[5]={1.1,2.2,3.3,4.4,5.5};
   double int_aver=AverageFromArray(a,5);
   double double_aver=AverageFromArray(b,5);
   Print("int_aver = ",int_aver,"   double_aver = ",double_aver);
  }
//--- Result of the script
// Calculate the average for an array of int type
// Calculate the average for an array of double type
// int_aver= 3.00000000    double_aver= 3.30000000
```

Function overloading is a process of creating several functions with the same name, but different parameters. This means that in overloaded variants of a function, the number of arguments and/or their type must be different. A specific function variant is selected based on the correspondence of the list of arguments when calling the function, to the list of parameters in the function declaration.

When an overloaded function is called, the compiler must have an algorithm to select the appropriate function. The algorithm that performs this choice depends on castings of what types are present. The best correspondence must be unique. An overloaded function must be the best match among all the other variants for at least one argument. At the same time it must match for all other arguments not worse than other variants.

Below is a matching algorithm for each argument.

Algorithm of Choosing an Overloaded Function

1. Use strict matching (if possible).
2. Try standard type increase.
3. Try standard typecasting.

The standard type increase is better than other standard conversions. Increase is the conversion of float to double, of bool, char, short or enum to int. Typecasting of arrays of similar integer types also belongs to typecasting. Similar types are: bool, char, uchar, since all the three types are single-byte integers; double-byte integers short and ushort; 4-byte integers int, uint, and color; long, ulong, and datetime.

Of course, the strict matching is the best. To achieve such a consistency typecasting can be used. The compiler cannot cope with ambiguous situations. Therefore you should not rely on subtle differences of types and implicit conversions that make the overloaded function unclear.

If you doubt, use explicit conversion to ensure strict compliance.

Examples of overloaded functions in MQL5 can be seen in the example of [ArrayInitialize()]() functions.

Function overloading rules apply to [overload of class methods.]()

Overloading of system functions is allowed, but it should be observed that the compiler is able to accurately select the necessary function. For example, we can overload the system function [MathMax()]() in 4 different ways, but only two variants are correct.

Example:

```
// 1. overload is allowed - function differs from built-in MathMax() function in the
double MathMax(double a,double b,double c);

// 2. overload is not allowed!
// number of parameters is different, but the last has a default value
// this leads to the concealment of the system function when calling, which is unaccep
double MathMax(double a,double b,double c=DBL_MIN);

// 3. overload is allowed - normal overload by type of parameters a and b
double MathMax(int a,int b);

// 4. overload is not allowed!
// the number and types of parameters are the same as in original double MathMax(doubl
int MathMax(double a,double b);
```

See also

[Overload](), [Virtual Functions](), [Polymorphism]()

Operation Overloading

For ease of code reading and writing, overloading of some operations is allowed. Overloading operator is written using the keyword operator. The following operators can be overloaded:

- binary +,-,/,*,%,<<,>>,==,!=,<,>,<=,>=,=,+=,-=,/=,*=,%=,&=,|=,^=,<<=,>>=,&&,||,&,|,^
- unary +,-,++,--,!,~
- assignment operator =
- indexing operator []

Operation overloading allows the use of the operating notation (written in the form of simple expressions) for complex objects - structures and classes. Writing expressions using overloaded operations simplifies the view of the source code, because a more complex implementation is hidden.

For example, consider complex numbers, which consist of real and imaginary parts. They are widely used in mathematics. The MQL5 language has no data type to represent complex numbers, but it is possible to create a new data type in the form of a structure or class. Declare the complex structure and define four methods that implement four arithmetic operations:

```
//+------------------------------------------------------------------+
//| A structure for operations with complex numbers                  |
//+------------------------------------------------------------------+
struct complex
  {
   double               re; // Real part
   double               im; // Imaginary part
   //--- Constructors
                     complex():re(0.0),im(0.0) {   }
                     complex(const double r):re(r),im(0.0) {   }
                     complex(const double r,const double i):re(r),im(i) {   }
                     complex(const complex &o):re(o.re),im(o.im) { }
   //--- Arithmetic operations
   complex           Add(const complex &l,const complex &r) const;  // Addition
   complex           Sub(const complex &l,const complex &r) const;  // Subtraction
   complex           Mul(const complex &l,const complex &r) const;  // Multiplication
   complex           Div(const complex &l,const complex &r) const;  // Division
  };
```

Now, in our code we can declare variables representing complex numbers, and work with them.

For example:

```
void OnStart()
  {
//--- Declare and initialize variables of a complex type
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- Sum up two numbers
   complex z;
```

```
    z=a.Add(a,b);
    PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- Multiply two numbers
    z=a.Mul(a,b);
    PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- Divide two numbers
    z=a.Div(a,b);
    PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
```

But it would be more convenient to use usual operators "+", "-", "*" and "/" for ordinary arithmetic operations with complex numbers.

Keyword operator is used for defining a member function that performs type conversion. Unary and binary operations for class object variables can be overloaded as non-static member functions. They implicitly act on the class object.

Most binary operations can be overloaded like regular functions that take one or both arguments as a class variable or a pointer to an object of this class. For our type complex, overloading in the declaration will look like this:

```
    //--- Operators
    complex operator+(const complex &r) const { return(Add(this,r)); }
    complex operator-(const complex &r) const { return(Sub(this,r)); }
    complex operator*(const complex &r) const { return(Mul(this,r)); }
    complex operator/(const complex &r) const { return(Div(this,r)); }
```

The full example of the script:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare and initialize variables of type complex
    complex a(2,4),b(-4,-2);
    PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
    //a.re=5;
    //a.im=1;
    //b.re=-1;
    //b.im=-5;
//--- Sum up two numbers
    complex z=a+b;
    PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- Multiply two numbers

    z=a*b;
    PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- Divide two numbers
    z=a/b;
```

```
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
//+------------------------------------------------------------------+
//| A structure for operations with complex numbers                  |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // Real part
   double            im; // Imaginary part
   //--- Constructors
                     complex():re(0.0),im(0.0) {  }
                     complex(const double r):re(r),im(0.0) {  }
                     complex(const double r,const double i):re(r),im(i) {  }
                     complex(const complex &o):re(o.re),im(o.im) { }
   //--- Arithmetic operations
   complex           Add(const complex &l,const complex &r) const;  // Addition
   complex           Sub(const complex &l,const complex &r) const;  // Subtraction
   complex           Mul(const complex &l,const complex &r) const;  // Multiplication
   complex           Div(const complex &l,const complex &r) const;  // Division
   //--- Binary operators
   complex operator+(const complex &r) const { return(Add(this,r)); }
   complex operator-(const complex &r) const { return(Sub(this,r)); }
   complex operator*(const complex &r) const { return(Mul(this,r)); }
   complex operator/(const complex &r) const { return(Div(this,r)); }
  };
//+------------------------------------------------------------------+
//| Addition                                                         |
//+------------------------------------------------------------------+
complex complex::Add(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re+r.re;
   res.im=l.im+r.im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Subtraction                                                      |
//+------------------------------------------------------------------+
complex complex::Sub(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re-r.re;
   res.im=l.im-r.im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Multiplication                                                   |
//+------------------------------------------------------------------+
complex complex::Mul(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re*r.re-l.im*r.im;
   res.im=l.re*r.im+l.im*r.re;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Division                                                         |
//+------------------------------------------------------------------+
complex complex::Div(const complex &l,const complex &r) const
  {
//--- Empty complex number
   complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- Check for zero
   if(r.re==0 && r.im==0)
     {
      Print(__FUNCTION__+": number is zero");
      return(res);
     }
//--- Auxiliary variables
   double e;
   double f;
//--- Selecting calculation variant
   if(MathAbs(r.im)<MathAbs(r.re))
     {
      e = r.im/r.re;
      f = r.re+r.im*e;
      res.re=(l.re+l.im*e)/f;
      res.im=(l.im-l.re*e)/f;
     }
   else
     {
      e = r.re/r.im;
      f = r.im+r.re*e;
      res.re=(l.im+l.re*e)/f;
      res.im=(-l.re+l.im*e)/f;
     }
//--- Result
   return res;
  }
```

Most unary operations for classes can be overloaded as ordinary functions that accept a single class object argument or a pointer to it. Add overloading of unary operations "-" and "!".

```
//+------------------------------------------------------------------+
//| A structure for operations with complex numbers                  |
//+------------------------------------------------------------------+
struct complex
  {
   double            re;       // Real part
   double            im;       // Imaginary part
...
   //--- Unary operators
   complex operator-()  const; // Unary minus
   bool    operator!()  const; // Negation
  };
...
//+------------------------------------------------------------------+
//| Overloading the "unary minus" operator                           |
//+------------------------------------------------------------------+
complex complex::operator-() const
  {
   complex res;
//---
   res.re=-re;
   res.im=-im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Overloading the "logical negation" operator                      |
//+------------------------------------------------------------------+
bool complex::operator!() const
  {
//--- Are the real and imaginary parts of the complex number equal to zero?
   return (re!=0 && im!=0);
  }
```

Now we can check the value of a complex number for zero and get a negative value:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare and initialize variables of type complex
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- Divide the two numbers
```

```
   complex z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//--- A complex number is equal to zero by default (in the default constructor re==0 a
   complex zero;
   Print("!zero=",!zero);
//--- Assign a negative value
   zero=-z;
   PrintFormat("z=%.2f+i*%.2f,  zero=%.2f+i*%.2f",z.re,z.im, zero.re,zero.im);
   PrintFormat("-zero=%.2f+i*%.2f",-zero.re,-zero.im);
//--- Check for zero once again
   Print("!zero=",!zero);
//---
  }
```

Note that we did not have to overload the assignment operator "=", as structures of simple types can be directly copied one into each other. Thus, we can now write a code for calculations involving complex numbers in the usual manner.

Overloading of the indexing operator allows to obtain the values of the arrays enclosed in an object, in a simple and familiar way, and it also contributes to a better readability of the source code. For example, we need to provide access to a symbol in the string at the specified position. A string in MQL5 is a separate type string, which is not an array of symbols, but with the help of an overloaded indexing operation we can provide a simple and transparent work in the generated CString class:

```
//+------------------------------------------------------------------+
//| Class to access symbols in string as in array of symbols         |
//+------------------------------------------------------------------+
class CString
  {
   string            m_string;

public:
                     CString(string str=NULL):m_string(str) { }
   ushort operator[] (int x) { return(StringGetCharacter(m_string,x)); }
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- An array for receiving symbols from a string
   int     x[]={ 19,4,18,19,27,14,15,4,17,0,19,14,17,27,26,28,27,5,14,
                 17,27,2,11,0,18,18,27,29,30,19,17,8,13,6 };
   CString str("abcdefghijklmnopqrstuvwxyz[ ]CS");
   string  res;
//--- Make up a phrase using symbols from the str variable
   for(int i=0,n=ArraySize(x);i<n;i++)
     {
      res+=ShortToString(str[x[i]]);
     }
//--- Show the result
   Print(res);
  }
```

Another example of overloading of the indexing operation is operations with matrices. The matrix represents a two-dimensional dynamic array, the array size is not defined in advance. Therefore, you cannot declare an array of form array[][] without specifying the size of the second dimension, and then pass this array as a parameter. A possible solution is a special class CMatrix, which contains an array of CRow class objects.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Operations of addition and multiplication of matrices
   CMatrix A(3),B(3),C();
//--- Prepare an array for rows
   double a1[3]={1,2,3}, a2[3]={2,3,1}, a3[3]={3,1,2};
   double b1[3]={3,2,1}, b2[3]={1,3,2}, b3[3]={2,1,3};
//--- Fill the matrices
   A[0]=a1; A[1]=a2; A[2]=a3;
   B[0]=b1; B[1]=b2; B[2]=b3;
//--- Output the matrices in the Experts log
   Print("---- Elements of matrix A");
   Print(A.String());
   Print("---- Elements of matrix B");
   Print(B.String());

//--- Addition of matrices
   Print("---- Addition of matrices A and B");
   C=A+B;
//--- Output the formatted string representation
   Print(C.String());

//--- Multiplication of matrices
   Print("---- Multiplication of matrices A and B");
   C=A*B;
   Print(C.String());

//--- Now we show how to get values in the style of dynamic arrays matrix[i][j]
   Print("Output the values of matrix C elementwise");
//--- Go through the matrix rows - CRow objects - in a loop
   for(int i=0;i<3;i++)
     {
      string com="| ";
      //--- Form rows from the matrix for the value
      for(int j=0;j<3;j++)
        {
         //--- Get the matrix element by the number of the row and column
         double element=C[i][j];// [i] - Access to CRow in the array m_rows[] ,
                                // [j] - Overloaded operator of indexing in CRow
         com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
        }
      com+="|";
      //--- Output the values of the row
      Print(com);
     }
  }
//+------------------------------------------------------------------+
//| Class "Row"                                                      |
//+------------------------------------------------------------------+
class CRow
  {
private:
   double             m_array[];
public:
   //--- Constructors and a destructor
                     CRow(void)          { ArrayResize(m_array,0);    }
                     CRow(const CRow &r) { this=r;                    }
                     CRow(const double &array[]);
                    ~CRow(void){};
   //--- Number of elements in the row
   int               Size(void) const    { return(ArraySize(m_array));}
   //--- Returns a string with values
   string            String(void) const;
   //--- Indexing operator
   double            operator[](int i) const  { return(m_array[i]);   }
   //--- Assignment operators
   void              operator=(const double  &array[]); // An array
   void              operator=(const CRow & r);         // Another CRow object
   double            operator*(const CRow &o);          // CRow object for multiplicat
  };
//+------------------------------------------------------------------+
//| Constructor for initializing a row with an array                 |
//+------------------------------------------------------------------+
void  CRow::CRow(const double &array[])
  {
   int size=ArraySize(array);
//--- If the array is not empty
   if(size>0)
     {
      ArrayResize(m_array,size);
      //--- Fill with values
      for(int i=0;i<size;i++)
         m_array[i]=array[i];
     }
//---
```

```
  }
//+------------------------------------------------------------------+
//| Assignment operation for the array                               |
//+------------------------------------------------------------------+
void CRow::operator=(const double &array[])
  {
   int size=ArraySize(array);
   if(size==0) return;
//--- Fill the array with values
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//| Assignment operation for CRow                                    |
//+------------------------------------------------------------------+
void CRow::operator=(const CRow  &r)
  {
   int size=r.Size();
   if(size==0) return;
//--- Fill the array with values
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=r[i];
//---
  }
//+------------------------------------------------------------------+
//| Operator of multiplication by another row                        |
//+------------------------------------------------------------------+
double CRow::operator*(const CRow &o)
  {
   double res=0;
//--- Verifications
   int size=Size();
   if(size!=o.Size() || size==0)
     {
      Print(__FUNCSIG__,": Failed to multiply two matrices, their sizes are different"
      return(res);
     }
//--- Multiply arrays elementwise and add the products
   for(int i=0;i<size;i++)
      res+=m_array[i]*o[i];
//--- Result
   return(res);
  }
//+------------------------------------------------------------------+
//| Returns a formatted string representation                        |
//+------------------------------------------------------------------+
string CRow::String(void) const
  {
   string out="";
//--- If the size of the array is greater than zero
   int size=ArraySize(m_array);
//--- We work only with a non-zero number of array elements
   if(size>0)
     {
      out="{";
      for(int i=0;i<size;i++)
        {
         //--- Collect the values to a string
         out+=StringFormat(" %G;",m_array[i]);
        }
      out+=" }";
     }
//--- Result
   return(out);
  }
//+------------------------------------------------------------------+
//| Class "Matrix"                                                   |
//+------------------------------------------------------------------+
class CMatrix
  {
private:
   CRow              m_rows[];

public:
   //--- Constructors and a destructor
                     CMatrix(void);
                     CMatrix(int rows)  { ArrayResize(m_rows,rows);             }
                    ~CMatrix(void){};
   //--- Get the matrix sizes
   int               Rows()       const { return(ArraySize(m_rows));            }
   int               Cols()       const { return(Rows()>0? m_rows[0].Size():0); }
   //--- Returns the value of the column in the form of a CRow row
   CRow              GetColumnAsRow(const int col_index) const;
   //--- Returns a string with the matrix values
   string            String(void) const;
   //--- The indexing operator returns a string by its number
   CRow *operator[](int i) const        { return(GetPointer(m_rows[i]));        }
   //--- Addition operator
   CMatrix           operator+(const CMatrix &m);
   //--- Multiplication operator
   CMatrix           operator*(const CMatrix &m);
   //--- Assignment operator
   CMatrix          *operator=(const CMatrix &m);
  };
//+------------------------------------------------------------------+
//| A default constructor, create an array of rows of zero size      |
//+------------------------------------------------------------------+
CMatrix::CMatrix(void)
  {
//--- The zero number of rows in the matrix
   ArrayResize(m_rows,0);
//---
  }
//+------------------------------------------------------------------+
//| Returns the column value in the form of CRow                     |
//+------------------------------------------------------------------+
CRow  CMatrix::GetColumnAsRow(const int col_index) const
  {
//--- A variable to get the values from the column
   CRow row();
//--- The number of rows in the matrix
   int rows=Rows();
//--- If the number of rows is greater than zero, execute the operation
   if(rows>0)
     {
      //--- An array to receive the values of the column with index col_index
      double array[];
      ArrayResize(array,rows);
      //--- Filling the array
      for(int i=0;i<rows;i++)
        {
         //--- Check the number of the column for row i - it may exceed the boundarie
         if(col_index>=this[i].Size())
           {
            Print(__FUNCSIG__,": Error! Column number ",col_index,"> row size ",i);
            break; // row will be uninitialized object
           }
         array[i]=this[i][col_index];
        }
      //--- Create a CRow row based on the array values
      row=array;
     }
//--- Result
   return(row);
  }
//+------------------------------------------------------------------+
//| Addition of two matrices                                         |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator+(const CMatrix &m)
  {
//--- The number of rows and columns in the passed matrix
   int cols=m.Cols();
   int rows=m.Rows();
//--- The matrix to receive the addition results
   CMatrix res(rows);
//--- The sizes of the matrix must match
```

```
   if(cols!=Cols() || rows!=Rows())
     {
      //--- Addition impossible
      Print(__FUNCSIG__,": Failed to add two matrices, their sizes are different");
      return(res);
     }
//--- Auxiliary array
   double arr[];
   ArrayResize(arr,cols);
//--- Go through rows to add
   for(int i=0;i<rows;i++)
     {
      //--- Write the results of addition of matrix strings in the array
      for(int k=0;k<cols;k++)
        {
         arr[k]=this[i][k]+m[i][k];
        }
      //--- Place the array to the matrix row
      res[i]=arr;
     }
//--- return the result of addition of matrices
   return(res);
  }
//+------------------------------------------------------------------+
//| Multiplication of two matrices                                   |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator*(const CMatrix &m)
  {
//--- Number of columns of the first matrix, number of rows passed in the matrix
   int cols1=Cols();
   int rows2=m.Rows();
   int rows1=Rows();
   int cols2=m.Cols();
//--- Matrix to receive the addition result
   CMatrix res(rows1);
//--- Matrices should be coordinated
   if(cols1!=rows2)
     {
      //--- Multiplication impossible
      Print(__FUNCSIG__,": Failed to multiply two matrices, format is not compatible "
            "- number of columns in the first factor should be equal to the number of
      return(res);
     }
//--- Auxiliary array
   double arr[];
   ArrayResize(arr,cols1);
//--- Fill the rows in the multiplication matrix
   for(int i=0;i<rows1;i++)// Go through rows
     {
      //--- Reset the receiving array
      ArrayInitialize(arr,0);
      //--- Go through elements in the row
      for(int k=0;k<cols1;k++)
        {
         //--- Take values of column k of the matrix m in the for of CRow
         CRow column=m.GetColumnAsRow(k);
         //--- Multiply two rows and write the result of scalar multiplication of vect
         arr[k]=this[i]*column;
        }
      //--- place array arr[] in the i-th row of the matrix
      res[i]=arr;
     }
//--- Return the product of two matrices
   return(res);
  }
//+------------------------------------------------------------------+
//| Assignment operation                                             |
//+------------------------------------------------------------------+
CMatrix *CMatrix::operator=(const CMatrix &m)
  {
//--- Find and set the number of rows
   int rows=m.Rows();
   ArrayResize(m_rows,rows);
//--- Fill our rows with the values of rows of  the passed matrix
   for(int i=0;i<rows;i++) this[i]=m[i];
//---
   return(GetPointer(this));
  }
//+------------------------------------------------------------------+
//| String representation of the matrix                              |
//+------------------------------------------------------------------+
string CMatrix::String(void) const
  {
   string out="";
   int rows=Rows();
//--- Form string by string
   for(int i=0;i<rows;i++)
     {
      out=out+this[i].String()+"\r\n";
     }
//--- Result
   return(out);
  }
```

See also

Overloading, Arithmetic Operations, Function Overloading, Precedence Rules

Description of External Functions

External functions defined in another module must be explicitly described. The description includes returned type, function name and series of input parameters with their types. The absence of such a description can lead to errors when compiling, building, or executing a program. When describing an external object, use the keyword *#import* indicating the module.

Examples:

```
#import "user32.dll"
  int     MessageBoxW(int hWnd ,string szText,string szCaption,int nType);
  int     SendMessageW(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex5"
  double  round(double value);
#import
```

With the help of import, it is easy to describe functions that are called from external DLL or compiled EX5 libraries. EX5 libraries are compiled ex5 files, which have the library property. Only function described with the export modifier can be imported from EX5 libraries.

Please keep in mind that DLL and EX5 libraries should have different names (regardless of the directories they are located in) if they are imported together. All imported functions have the scope resolution corresponding to the library's "file name".

Example:

```
#import "kernel32.dll"
   int GetLastError();
#import "lib.ex5"
   int GetLastError();
#import

class CFoo
  {
public:
   int               GetLastError() { return(12345); }
   void              func()
     {
      Print(GetLastError());           // call of the class method
      Print(::GetLastError());         // call of the MQL5 function
      Print(kernel32::GetLastError()); // call of the DLL library function from kerne
      Print(lib::GetLastError());      // call of the EX5 library function from lib.e
     }
  };

void OnStart()
  {
   CFoo foo;
   foo.func();
  }
```

See also

Overload, Virtual Functions, Polymorphism

Exporting Functions

A function declared in a mql5 program with the *export* postmodifier can be used in another mql5 program. Such a function is called exportable, and it can be called from other programs after compilation.

```
int Function() export
  {
  }
```

This modifier orders the compiler to add the function into the table of EX5 functions exported by this ex5 file. Only function with such a modifier are accessible ("visible") from other mql5 programs.

The library property tells the compiler that the EX5-file will be a library, and the compiler will show it in the header of EX5.

All functions that are planned as exportable ones must be marked with the *export* modifier.

See also

Overload, Virtual Functions, Polymorphism

Event Handling Functions

The MQL5 language provides processing of some predefined events. Functions for handling these events must be defined in a MQL5 program; function name, return type, composition of parameters (if there are any) and their types must strictly conform to the description of the event handler function.

The event handler of the client terminal identifies functions, handling this or that event, by the type of return value and type of parameters. If other parameters, not corresponding to below descriptions, are specified for a corresponding function, or another return type is indicated for it, such a function will not be used as an event handler.

OnStart

The OnStart() function is the Start event handler, which is automatically generated **only** for running **scripts**. It must be of void type, with no parameters:

```
void OnStart();
```

For the OnStart() function, the int return type can be specified.

OnInit

The OnInit() function is the Init event handler. It must be of void or int type, with no parameters:

```
void OnInit();
```

The Init event is generated immediately after an Expert Advisor or an indicator is downloaded; this event is not generated for scripts. The OnInit() function is used for initialization. If OnInit() has the int type of the return value, the non-zero return code means unsuccessful initialization, and it generates the Deinit event with the code of deinitialization reason REASON_INITFAILED.

When returning INIT_FAILED, the EA is forcibly unloaded from the chart.

When returning INIT_FAILED, the indicator is not unloaded from the chart. The indicator remaining on the chart is non-operational — event handlers are not called in the indicator.

To optimize input parameters of an Expert Advisor, it is recommended to use values of the ENUM_INIT_RETCODE enumeration as the return code. These values are used for organizing the course of optimization, including the selection of the most appropriate testing agents. During initialization of an Expert Advisor before the start of testing you can request information about the configuration and resources of an agent (the number of cores, amount of free memory, etc.) using the TerminalInfoInteger() function. Based on the information obtained, you can either allow to use this testing agent, or reject using it during the optimization of this Expert Advisor.

ENUM_INIT_RETCODE

Identifier	Description
INIT_SUCCEEDED	Successful initialization, testing of the Expert Advisor can be continued. This code means the same as a null value – the Expert Advisor has been successfully initialized in the tester.

Identifier	Description
INIT_FAILED	Initialization failed; there is no point in continuing testing because of fatal errors. For example, failed to create an indicator that is required for the work of the Expert Advisor. This return value means the same as a value other than zero - initialization of the Expert Advisor in the tester failed.
INIT_PARAMETERS_INCORRECT	This value means the incorrect set of input parameters. The result string containing this return code is highlighted in red in the general optimization table. Testing for the given set of parameters of the Expert Advisor will not be executed, the agent is free to receive a new task. Upon receiving this value, the strategy tester will reliably not pass this task to other agents for retry.
INIT_AGENT_NOT_SUITABLE	No errors during initialization, but for some reason the agent is not suitable for testing. For example, not enough memory, no OpenCL support, etc. After the return of this code, the agent will not receive tasks until the end of this optimization.

The OnInit() function of the void type always denotes successful initialization.

OnDeinit

The OnDeinit() function is called during deinitialization and is the Deinit event handler. It must be declared as the void type and should have one parameter of the const int type, which contains the code of deinitialization reason. If a different type is declared, the compiler will generate a warning, but the function will not be called. For scripts the Deinit event is not generated and therefore the OnDeinit() function can't be used in scripts.

```
void OnDeinit(const int reason);
```

The Deinit event is generated for Expert Advisors and indicators in the following cases:

- before reinitialization due to the change of a symbol or chart period, to which the mql5 program is attached;
- before reinitialization due to the change of input parameters;
- before unloading the mql5 program.

OnTick

The NewTick event is generated for **Expert Advisors only** when a new tick for a symbol is received, to the chart of which the Expert Advisor is attached. It's useless to define the OnTick() function in a custom indicator or script, because the NewTick event is not generated for them.

The Tick event is generated only for Expert Advisors, but this does not mean that Expert Advisors required the OnTick() function, since not only NewTick events are generated for Expert Advisors, but also events of Timer, BookEvent and ChartEvent are generated. It must be declared as the void type, with no parameters:

```
void OnTick();
```

OnTimer

The OnTimer() function is called when the Timer event occurs, which is generated by the system timer only for Expert Advisors and indicators - it can't be used in scripts. The frequency of the event occurrence is set when subscribing to notifications about this event to be received by the EventSetTimer() function.

You can unsubscribe from receiving timer events for a particular Expert Advisor using the EventKillTimer() function. The function must be defined with the void type, with no parameters:

```
void OnTimer();
```

It is recommended to call the EventSetTimer() function once in the OnInit() function, and the EventKillTimer() function should be called once in OnDeinit().

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as the mql5 program stops operating, the timer is destroyed forcibly, if it was created but hasn't been disabled by the EventKillTimer() function.

OnTrade

The function is called when the Trade event occurs, which appears when you change the list of placed orders and open positions, the history of orders and history of deals. When a trade activity is performed (pending order opening, position opening/closing, stops setting, pending order triggering, etc.) the history of orders and deals and/or list of positions and current orders is changed accordingly.

```
void OnTrade();
```

Users must independently implement in the code the verification of a trade account state when such an event is received (if this is required by the trade strategy conditions). If the OrderSend() function call has been completed successfully and returned a value of true, this means that the trading server has put the order into the queue for execution and assigned a ticket number to it. As soon as the server processes this order, the Trade event will be generated. And if a user remembers the ticket value, he/she will be able to find out what happened to the order using this value during OnTrade() event handling.

OnTradeTransaction

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using OrderSend and OrderSendAsync functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;

- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions. Arrival of such a transaction at the terminal is a TradeTransaction event. It calls OnTradeTransaction handler

```
void  OnTradeTransaction(
   const MqlTradeTransaction&    trans,        // trade transaction structure
   const MqlTradeRequest&        request,      // request structure
   const MqlTradeResult&         result        // result structure
   );
```

The handler contains three parameters:

- **trans** - this parameter gets MqlTradeTransaction structure describing a trade transaction applied to a trade account;
- **request** - this parameter gets MqlTradeRequest structure describing a trade request;
- **result** - this parameter gets MqlTradeResult structure describing a trade request execution result.

The last two **request** and **result** parameters are filled by values only for TRADE_TRANSACTION_REQUEST type transaction, data on transaction can be received from *type* parameter of **trans** variable. Note that in this case, *request_id* field in **result** variable contains ID of **request** trade request, after the execution of which the trade transaction described in **trans** variable has been performed. Request ID allows to associate the performed action (OrderSend or OrderSendAsync functions call) with the result of this action sent to OnTradeTransaction().

One trade request manually sent from the terminal or via OrderSend()/OrderSendAsync() functions can generate several consecutive transactions on the trade server. Priority of these transactions' arrival at the terminal is not guaranteed. Thus, you should not expect that one group of transactions will arrive after another one when developing your trading algorithm.

- All types of trade transactions are described in ENUM_TRADE_TRANSACTION_TYPE enumeration.
- MqlTradeTransaction structure describing a trade transaction is filled in different ways depending on a transaction type. For example, only type field (trade transaction type) must be analyzed for TRADE_TRANSACTION_REQUEST type transactions. The second and third parameters of OnTradeTransaction function (request and result) must be analyzed for additional data. For more information, see "Structure of a Trade Transaction".
- A trade transaction description does not deliver all available information concerning orders, deals and positions (e.g., comments). OrderGet*, HistoryOrderGet*, HistoryDealGet* and PositionGet* functions should be used to get extended information.

After applying trade transactions for a client account, they are consistently placed to the terminal trade transactions queue, from which they consistently sent to OnTradeTransaction entry point in order of arrival at the terminal.

When handling trade transactions by an Expert Advisor using OnTradeTransaction handler, the terminal continues handling newly arrived trade transactions. Therefore, the state of a trade account can change during OnTradeTransaction operation already. For example, while an MQL5 program handles an event of adding a new order, it may be executed, deleted from the list of the open ones and moved to the history. Further on, the application will be notified of these events.

Transactions queue length comprises 1024 elements. If OnTradeTransaction handles a new transaction for too long, the old ones in the queue may be superseded by the newer ones.

- Generally, there is no accurate ratio of the number of OnTrade and OnTradeTransaction calls. One OnTrade call corresponds to one or several OnTradeTransaction calls.
- OnTrade is called after appropriate OnTradeTransaction calls.

OnTester

The OnTester() function is the handler of the Tester event that is automatically generated after a history testing of an Expert Advisor on the chosen interval is over. The function must be defined with the double type, with no parameters:

```
double OnTester();
```

The function is called right before the call of OnDeinit() and has the same type of the return value - double. OnTester() can be used only in the testing of Expert Advisors. Its main purpose is to calculate a certain value that is used as the Custom max criterion in the genetic optimization of input parameters.

In the genetic optimization descending sorting is applied to results within one generation. I.e. from the point of view of the optimization criterion, the best results are those with largest values (for the Custom max optimization criterion values returned by the OnTester function are taken into account). In such a sorting, the worst values are positioned at the end and further thrown off and do not participate in the forming of the next generation.

OnTesterInit

The function is called in EAs when the TesterInit event occurs to perform necessary actions before optimization in the strategy tester. There are two function types.

The **version that returns the result**

```
int  OnTesterInit(void);
```

Return Value

int type value, zero means successful initialization of an EA launched on a chart before optimization starts.

The OnTesterInit() call that returns the execution result is recommended for use since it not only allows for program initialization, but also returns an error code in case of an early optimization stop. Return of any value other than INIT_SUCCEEDED (0) means an error, no optimization is launched.

The version without a result return is left only for compatibility with old codes. Not recommended for use

```
void  OnTesterInit(void);
```

With the start of optimization, an Expert Advisor with the OnTesterDeinit() or OnTesterPass() handler is automatically loaded in a separate terminal chart with the symbol and period specified in the tester, and receives the TesterInit event. The function is used for Expert Advisor initialization before the start of optimization for further processing of optimization results.

OnTesterPass

The OnTesterPass() function is the handler of the TesterPass event, which is automatically generated when a frame is received during Expert Advisor optimization in the strategy tester. The function must be defined with the void type. It has no parameters:

```
void OnTesterPass();
```

An Expert Advisor with the OnTesterPass() handler is automatically loaded in a separate terminal chart with the symbol/period specified for testing, and gets TesterPass events when a frame is received during optimization. The function is used for dynamic handling of optimization results "on the spot" without waiting for its completion. Frames are added using the FrameAdd() function, which can be called after the end of a single pass in the OnTester() handler.

OnTesterDeinit

OnTesterDeinit() is the handler of the TesterDeinit event, which is automatically generated after the end of Expert Advisor optimization in the strategy tester. The function must be defined with the void type. It has no parameters:

```
void OnTesterDeinit();
```

An Expert Advisor with the TesterDeinit() handler is automatically loaded on a chart at the start of optimization, and receives TesterDeinit after its completion. The function is used for final processing of all optimization results.

OnBookEvent

The OnBookEvent() function is the BookEvent handler. BookEvent is generated for Expert Advisors and indicators when Depth of Market changes. It must be of the void type and have one parameter of the string type:

```
void OnBookEvent (const string& symbol);
```

To receive BookEvent events for any symbol, you just need to pre-subscribe to receive these events for this symbol using the MarketBookAdd() function. In order to unsubscribe from receiving the BookEvent events for a particular symbol, call MarketBookRelease().

Unlike other events, the BookEvent event is broadcast. This means that if one Expert Advisor subscribes to receiving BookEvent events using MarketBookAdd, all the other Experts Advisors that have the OnBookEvent() handler will receive this event. It is therefore necessary to analyze the name of the symbol, which is passed to the handler as the *const string& symbol* parameter.

OnChartEvent

OnChartEvent() is the handler of a group of ChartEvent events:

- CHARTEVENT_KEYDOWN — event of a keystroke, when the chart window is focused;
- CHARTEVENT_MOUSE_MOVE — mouse move events and mouse click events (if CHART_EVENT_MOUSE_MOVE=true is set for the chart);
- CHARTEVENT_OBJECT_CREATE — event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart);

- CHARTEVENT_OBJECT_CHANGE – event of change of an object property via the properties dialog;
- CHARTEVENT_OBJECT_DELETE – event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart);
- CHARTEVENT_CLICK – event of a mouse click on the chart;
- CHARTEVENT_OBJECT_CLICK – event of a mouse click in a graphical object belonging to the chart;
- CHARTEVENT_OBJECT_DRAG – event of a graphical object move using the mouse;
- CHARTEVENT_OBJECT_ENDEDIT – event of the finished text editing in the entry box of the LabelEdit graphical object;
- CHARTEVENT_CHART_CHANGE – event of chart changes;
- CHARTEVENT_CUSTOM+n – ID of the user event, where n is in the range from 0 to 65535.
- CHARTEVENT_CUSTOM_LAST – the last acceptable ID of a custom event (CHARTEVENT_CUSTOM +65535).

The function can be called only in Expert Advisors and indicators. The function should be of void type with 4 parameters:

```
void OnChartEvent(const int id,          // Event ID
                  const long& lparam,    // Parameter of type long event
                  const double& dparam,  // Parameter of type double event
                  const string& sparam   // Parameter of type string events
  );
```

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through these parameters are listed in the table below.

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of times the keystroke is repeated as a result of the user holding down the key)	The string value of a bit mask describing the status of keyboard buttons
Mouse events (if property CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
Event of graphical object creation (if CHART_EVENT_O	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
BJECT_CREATE=true is set for the chart)				
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit graphical object, in which text editing has completed
Event of chart Changes	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

OnCalculate

The OnCalculate() function is called only in custom indicators when it's necessary to calculate the indicator values by the Calculate event. This usually happens when a new tick is received for the symbol, for which the indicator is calculated. This indicator is not required to be attached to any price chart of this symbol.

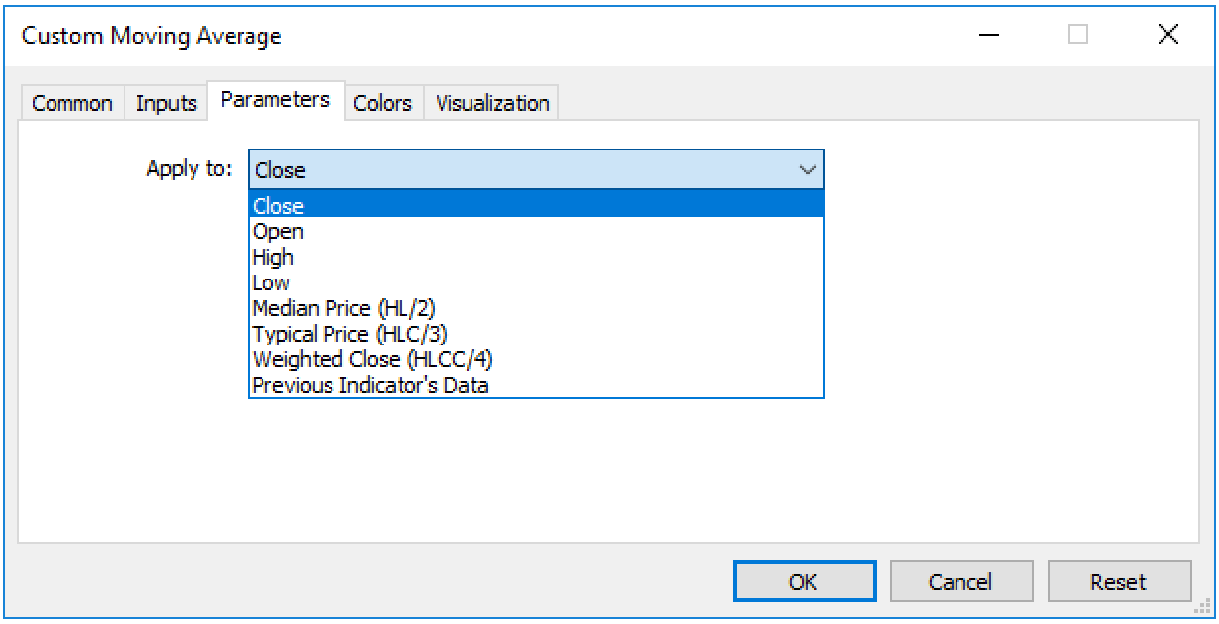
The OnCalculate() function must have a return type int. There are two possible definitions. Within one indicator you cannot use both versions of the function.

The first form is intended for those indicators that can be calculated on a single data buffer. An example of such an indicator is Custom Moving Average.

```
int OnCalculate (const int rates_total,      // size of the price[] array
                 const int prev_calculated,  // bars handled on a previous call
                 const int begin,            // where the significant data start from
                 const double& price[]       // array to calculate
   );
```

As the price[] array, one of timeseries or a calculated buffer of some indicator can be passed. To determine the direction of indexing in the price[] array, call ArrayGetAsSeries(). In order not to depend on the default values, you must unconditionally call the ArraySetAsSeries() function for those arrays, that are expected to work with.

Necessary time series or an indicator to be used as the price[] array can be selected by the user in the "Parameters" tab when starting the indicator. To do this, you should specify the necessary item in the drop-down list of "Apply to" field.


Custom Moving Average
Common
Inputs
Parameters
Colors
Visualization
Apply to:
Close
Close
Open
High
Low
Median Price (HL/2)
Typical Price (HLC/3)
Weighted Close (HLCC/4)
Previous Indicator's Data
OK
Cancel
Reset

To receive values of a custom indicator from other mql5 programs, the iCustom() function is used, which returns the indicator handle for subsequent operations. You can also specify the appropriate price[] array or the handle of another indicator. This parameter should be transmitted last in the list of input variables of the custom indicator.

Example:

```
void OnStart()
  {
//---
   string terminal_path=TerminalInfoString(STATUS_TERMINAL_PATH);
   int handle_customMA=iCustom(Symbol(),PERIOD_CURRENT, "Custom Moving Average",13,0,
   if(handle_customMA>0)
      Print("handle_customMA = ",handle_customMA);
   else
      Print("Cannot open or not EX5 file '"+terminal_path+"\\MQL5\\Indicators\\"+"Cust
  }
```

In this example, the last parameter passed is the PRICE_TYPICAL value (from the ENUM_APPLIED_PRICE enumeration), which indicates that the custom indicator will be built on typical prices obtained as (High+Low+Close)/3. If this parameter is not specified, the indicator is built based on PRICE_CLOSE values, i.e. closing prices of each bar.

Another example that shows passing of the indicator handler as the last parameter to specify the price[] array, is given in the description of the iCustom() function.

The second form is intended for all other indicators, in which more than one time series is used for calculations.

```
int OnCalculate (const int rates_total,      // size of input time series
                 const int prev_calculated,  // bars handled in previous call
                 const datetime& time[],     // Time
                 const double& open[],       // Open
                 const double& high[],       // High
                 const double& low[],        // Low
                 const double& close[],      // Close
                 const long& tick_volume[],  // Tick Volume
                 const long& volume[],       // Real Volume
                 const int& spread[]         // Spread
   );
```

Parameters of open[], high[], low[] and close[] contain arrays with open prices, high and low prices and close prices of the current time frame. The time[] parameter contains an array with open time values, the spread[] parameter has an array containing the history of spreads (if any spread is provided for the traded security). The parameters of volume[] and tick_volume[] contain the history of trade and tick volume, respectively.

To determine the indexing direction of time[], open[], high[], low[], close[], tick_volume[], volume[] and spread[], call ArrayGetAsSeries(). In order not to depend on default values, you should unconditionally call the ArraySetAsSeries() function for those arrays, which are expected to work with.

The first rates_total parameter contains the number of bars, available to the indicator for calculation, and corresponds to the number of bars available in the chart.

We should note the connection between the return value of OnCalculate() and the second input parameter prev_calculated. During the function call, the prev_calculated parameter contains a value **returned** by OnCalculate() during **previous** call. This allows for economical algorithms for calculating the custom indicator in order to avoid repeated calculations for those bars that haven't changed since the previous run of this function.

For this, it is usually enough to return the value of the rates_total parameter, which contains the number of bars in the current function call. If since the last call of OnCalculate() price data has changed (a deeper history downloaded or history blanks filled), the value of the input parameter prev_calculated will be set to zero by the terminal.

Note: if OnCalculate returns zero, then the indicator values are not shown in the DataWindow of the client terminal.

To understand it better, it would be useful to start the indicator, which code is attached below.

Indicator Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Line
#property indicator_label1  "Line"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDarkBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         LineBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime& time[],
                const double& open[],
                const double& high[],
                const double& low[],
                const double& close[],
                const long& tick_volume[],
                const long& volume[],
                const int& spread[])
  {
//--- Get the number of bars available for the current symbol and chart period
   int bars=Bars(Symbol(),0);
   Print("Bars = ",bars,", rates_total = ",rates_total,",  prev_calculated = ",prev_ca
   Print("time[0] = ",time[0]," time[rates_total-1] = ",time[rates_total-1]);
//--- return value of prev_calculated for next call
```

```
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

See also

Running Programs, Client Terminal Events, Working with Events

Variables

Declaring Variables

Variables must be declared before they are used. Unique names are used to identify variables. To declare a variable, you must specify its type and a unique name. Declaration of variable is not an operator.

Simple types are:

- char, short, int, long, uchar, ushort, uint, ulong – integers;
- color – integer representing the RGB-color;
- datetime – the date and time, an unsigned integer containing the number of seconds since 0 hour January 1, 1970;
- bool – boolean values *true* and *false*;
- double – double-precision floating point number;
- float – single-precision floating point number;
- string – character strings.

Examples:

```
string szInfoBox;
int    nOrders;
double dSymbolPrice;
bool   bLog;
datetime tBegin_Data   = D'2004.01.01 00:00';
color    cModify_Color = C'0x44,0xB9,0xE6';
```

Complex or compound types:

Structures are composite data types, constructed using other types.

```
struct MyTime
  {
   int hour;    // 0-23
   int minute;  // 0-59
   int second;  // 0-59
  };
...
MyTime strTime; // Variable of the previously declared structure MyTime
```

You can't declare variables of the structure type until you declare the structure.

Arrays

Array is the indexed sequence of identical-type data:

```
int    a[50];       // One-dimensional array of 50 integers.
double m[7][50];    // Two-dimensional array of seven arrays,
                    // each of them consisting of 50 numbers.
MyTime t[100];      // Array containing elements such as MyTime
```

Only an integer can be an array index. No more than four-dimensional arrays are allowed. Numbering of array elements starts with 0. The last element of a one-dimensional array has the number which is 1 less than the array size. This means that call for the last element of an array consisting of 50 integers will appear as a[49]. The same concerns multidimensional arrays: A dimension is indexed from 0 to the dimension size-1. The last element of a two-dimensional array from the example will appear as m[6][49].

Static arrays can't be represented as timeseries, i.e., the ArraySetAsSeries() function, which sets access to array elements from the end to beginning, can't be applied to them. If you want to provide access to an array the same as in timeseries, use the dynamic array object.

If there is an attempt to access out of the array range, the executing subsystem will generate a critical error and the program will be stopped.

Built-in methods for working with arrays

The functions from the Array Functions section, as well as the built-in methods can be used to handle the arrays:

Method	Analog	Description
void array.Fill(const scalar value, const int start_pos=0, const int count=-1);	ArrayFill, ArrayInitialize	Fills the array with the specified value
void array.Free();	ArrayFree	Releases the dynamic array buffer and sets the zero dimension size to 0 (zero)
int array.Resize(const int range0_size, const int reserve); int array.Resize(const int range_sizes[], const int reserve);	ArrayResize	Sets a new size in the array first dimension
int array.Print();	ArrayPrint	Displays simple type array values in the journal
int array.Size(const int range=-1);	ArraySize, ArrayRange	Returns the number of elements in the entire array (range=-1) or in the specified array dimension
bool array.IsDynamic();	ArrayIsDynamic	Checks if the array is dynamic
bool array.IsIndicatorBuffer();		Checks if the array is an indicator buffer
bool array.IsSeries();	ArrayIsSeries	Checks if the array is a timeseries
bool array.AsSeries();	ArrayGetAsSeries	Checks the array indexing direction
bool array.AsSeries(const bool as_series);	ArraySetAsSeries	Sets the indexing direction in the array
int array.Copy(const src_array[], const int dst_start, const int	ArrayCopy	Copies array values to another array

Method	Analog	Description
src_start, const int cnt);		
int array.Compare(const src_array[], const int dst_start, const int src_start, const int cnt);	ArrayCompare	Returns the result of comparing two simple type arrays or custom structures
int array.Insert(const src_array[], const int dst_start, const int src_start, const int cnt);	ArrayInsert	Inserts the specified number of elements from a source array to a receiving one starting from a specified index
int array.Remove(const int start_pos, const int count);	ArrayRemove	Removes the specified number of elements from the array starting with a specified index
int array.Reverse(const int start_pos, const int count);	ArrayReverse	Reverses the specified number of elements in the array starting with a specified index
bool array.Swap(array& arr[]);	ArraySwap	Exchanges the content with another dynamic array of the same type
void array.Sort(sort_function);	ArraySort	Sorts numeric arrays by the first dimension
int array.Search(scalar value, search_function);	ArrayBsearch	Returns the index of the first element detected in the array first dimension
int array.Find((scalar value, search_function);		Performs a search in the array using the passed function and returns the index of the first detected element
array array.Select(scalar value, search_function);		Performs a search in the array using the passed function and returns the array with all detected elements

Access Specifiers

Access specifiers define how the compiler can access variables, members of structures or classes.

The const specifier declares a variable as a constant, and does not allow to change this variable during runtime. A single initialization of a variable is allowed when declaring it.

Example:

```
int OnCalculate (const int rates_total,      // size of the price[] array
                 const int prev_calculated,  // bars handled on a previous call
                 const int begin,            // where the significant data start from
                 const double& price[]       // array to calculate
   );
```

To access members of structures and classes use the following qualifiers:

- public – allows unrestricted access to the variable or class method
- protected – allows access from methods of this class, as well as from methods of publicly inherited classes. Other access is impossible;
- private – allows access to variables and class methods only from methods of the same class.
- virtual – applies only to class methods (but not to methods of structures) and tells the compiler that this method should be placed in the table of virtual functions of the class.

Storage Classes

There are three storage classes: static, input and extern. These modifiers of a storage class explicitly indicate to the compiler that corresponding variables are distributed in a pre-allocated area of memory, which is called the global pool. Besides, these modifiers indicate the special processing of variable data. If a variable declared on a local level is not a static one, memory for such a variable is allocated automatically at a program stack. Freeing of memory allocated for a non-static array is also performed automatically when going beyond the visibility area of the block, in which the array is declared.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects, Static Members of a Class

Local Variables

A variable declared inside a function is local. The scope of a local variable is limited to the function range inside which it is declared. Local variable can be initialized by outcome of any expression. Every call of the function initializes a local variable. Local variables are stored in memory area of the corresponding function.

Example:

```
int somefunc()
  {
   int ret_code=0;
   ...
   return(ret_code);
  }
```

Scope of a variable is a program part, in which a variable can be referred to. Variables declared inside a block (at the internal level), have the block as their scope. The block scope start with the variable declaration and ends with the final right brace.

Local variables declared in the beginning of a function also have the scope of block, as well as function parameters that are local variables. Any block can contain variable declarations. If blocks are nested and the identifier in the external block has the same name as the identifier in the internal block, the external block identifier is hidden, until the operation of the internal block is over.

Example:

```
void OnStart()
  {
//---
   int i=5;      // local variable of the function
     {
      int i=10;  // function variable
      Print("Inside block i = ",i); // result is  i=10;
     }
   Print("Outside block i = ",i);  // result is  i=5;
  }
```

This means that while the internal block is running, it sees values of its own local identifiers, not the values of identifiers with identical names in the external block.

Example:

```
void OnStart()
  {
//---
   int i=5;      // local variable of the function
   for(int i=0;i<3;i++)
      Print("Inside for i = ",i);
   Print("Outside the block i = ",i);
  }
/* Execution result
```

```
Inside for i = 0
Inside for i = 1
Inside for i = 2
Outside block i = 5
*/
```

Local variables declared as static have the scope of the block, despite the fact that they exist since the program start.

Stack

In every MQL5 program, a special memory area called stack is allocated for storing local function variables that are created automatically. One stack is allocated for all functions, its default size for indicators is equal to 1Mb. In Expert Advisors and scripts, stack size can be managed using the #property stacksize compiler directive (which sets the stack size in bytes), a memory of 8Mb is allocated by default for the stack.

Static local variables are stored in the same place where other static and global variables are stored - in a special memory area, which exists separately from the stack. Dynamically created variables also use a memory area separate from the stack.

With each function call, a place on the stack is allocated for internal non-static variables. After exiting the function, the memory is available for use again.

If from the first function the second one is called, then the second function occupies the required size from the remaining stack memory for its variables. Thus, when using included functions, stack memory will be sequentially occupied for each function. This may lead to a shortage of memory during one of the function calls, such a situation is called stack overflow.

Therefore, for large local data you should better use dynamic memory - when entering a function, allocate the memory, which is required for local needs, in the system (new, ArrayResize()), and when exiting the function, release the memory (delete, ArrayFree()).

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Formal Parameters

Parameters passed to the function are local. The scope is the function block. Formal parameters must have names differing from those of external variables and local variables defined within one function. Some values can be assigned to formal parameters in the function block. If a formal parameter is declared with the const modifier, its value can't be changed within the function.

Example:

```
void func(const int & x[], double y, bool z)
  {
   if(y>0.0 && !z)
      Print(x[0]);
   ...
  }
```

Formal parameters can be initialized by constants. In this case, the initializing value is considered as the default value. Parameters, next to the initialized one, must also be initialized.

Example:

```
void func(int x, double y = 0.0, bool z = true)
  {
   ...
  }
```

When calling such a function, the initialized parameters can be omitted, the defaults being substituted instead of them.

Example:

```
func(123, 0.5);
```

Parameters of simple types are passed by value, i.e., modifications of the corresponding local variable of this type inside the called function will not be reflected in the calling function. Arrays of any type and data of the structure type are always passed by reference. If it is necessary to prohibit modifying the array or structure contents, the parameters of these types must be declared with the *const* keyword.

There is an opportunity to pass parameters of simple types by reference. In this case, modification of such parameters inside the calling function will affect the corresponding variables passed by reference. In order to indicate that a parameter is passed by reference, put the & modifier after the data type.

Example:

```
void func(int& x, double& y, double & z[])
  {
   double calculated_tp;
   ...
   for(int i=0; i<OrdersTotal(); i++)
     {
      if(i==ArraySize(z))       break;
      if(OrderSelect(i)==false) break;
```

```
       z[i]=OrderOpenPrice();
      }
    x=i;
    y=calculated_tp;
   }
```

Parameters passed by reference can't be initialized by default values.

Maximum 64 parameters can be passed into a function.

See also

Input Variables, Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Static Variables

The storage class of static defines a static variable. The static modifier is indicated before the data type.

Example:

```
int somefunc()
  {
   static int flag=10;
   ...
   return(flag);
  }
```

A static variable can be initialized by a constant or constant expression corresponding to its type, unlike a simple local variable, which can be initialized by any expression.

Static variables exist from the moment of program execution and are initialized only once before the specialized functions OnInit() is called. If the initial values are not specified, variables of the static storage class are taking zero initial values.

Local variables declared with the *static* keyword retain their values throughout the function lifetime. With each next function call, such local variables contain the values that they had during the previous call.

Any variables in a block, except formal parameters of a function, can be defined as static. If a variable declared on a local level is not a static one, memory for such a variable is allocated automatically at a program stack.

Example:

```
int Counter()
  {
   static int count;
   count++;
   if(count%100==0) Print("Function Counter has been called ",count," times");
   return count;
  }
void OnStart()
  {
//---
   int c=345;
   for(int i=0;i<1000;i++)
     {
      int c=Counter();
     }
   Print("c =",c);
  }
```

See also

Data Types, Encapsulation and Extensibility of Types, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects, Static Class Members

Global Variables

Global variables are created by placing their declarations outside function descriptions. Global variables are defined at the same level as functions, i.e., they are not local in any block.

Example:

```
int GlobalFlag=10;   // Global variable
int OnStart()
  {
   ...
  }
```

The scope of global variables is the entire program. Global variables are accessible from all functions defined in the program. They are initialized to zero unless another initial value is explicitly defined. A global variable can be initialized only by a constant or constant expression that corresponds to its type.

Global variables are initialized only once after the program is loaded into the client terminal memory and before the first handling of the Init event. For global variables representing class objects, during their initialization the corresponding constructors are called. In scripts global variables are initialized before handling the Start event.

Note: Variables declared at global level must not be mixed up with the client terminal global variables that can be accessed using the GlobalVariable...() functions.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Input Variables

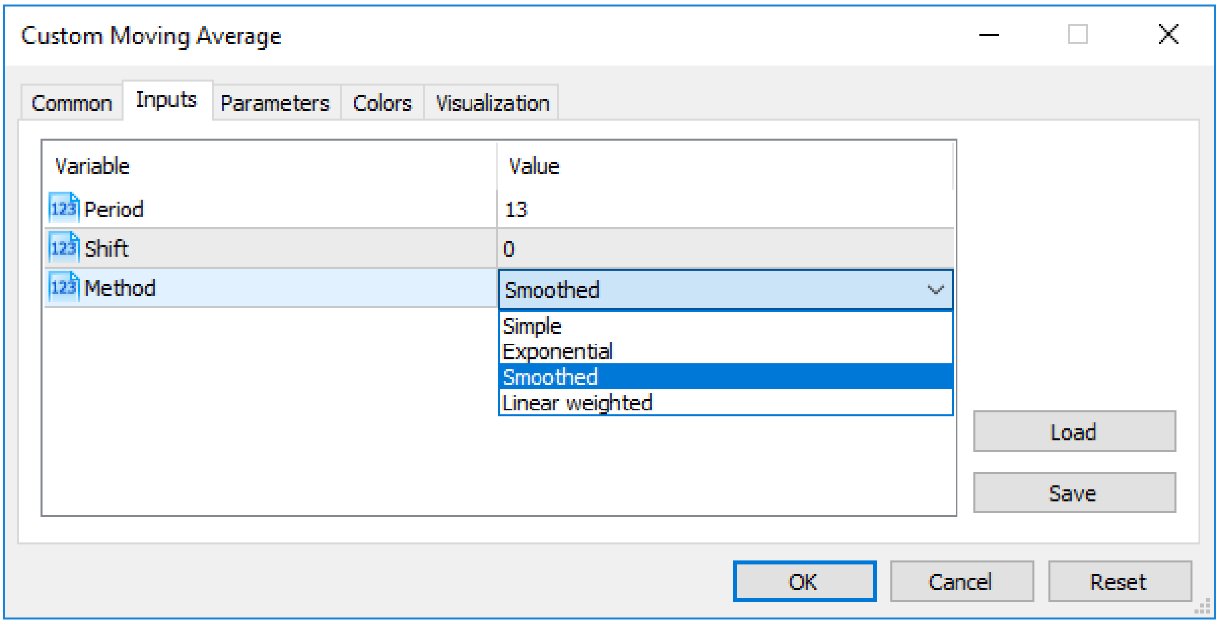
The input storage class defines the external variable. The *input* modifier is indicated before the data type. A variable with the input modifier can't be changed inside mql5-programs, such variables can be accessed for reading only. Values of input variables can be changed only by a user from the program properties window. External variables are always reinitialized immediately before the OnInit() is called.

The maximum length of input variable names is 63 characters. For the input parameter of string type, the **maximum** value length (string length) can be from 191 to 253 characters (see the Note). The minimum length is 0 characters (the value is not set).

Example:

```
//--- input parameters
input int             MA_Period=13;
input int             MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMMA;
```

Input variables determine the input parameters of a program. They are available from the Properties window of a program.

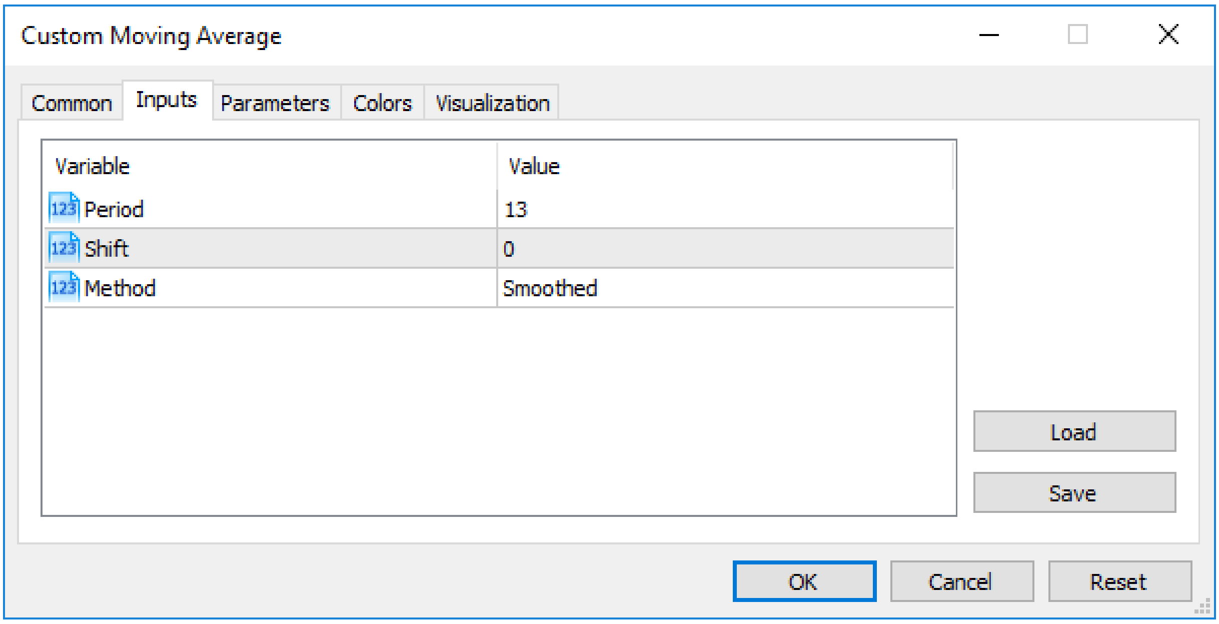


Variable	Value
Period	13
Shift	0
Method	Smoothed

There is another way to set how your input parameter will look like in the Inputs tab. For this, place a string comment after the description of an input parameter in the same line. In this way you can make names of input parameters more understandable for users.

Example:

```
//--- input parameters
input int             InpMAPeriod=13;          // Smoothing period
input int             InpMAShift=0;            // Line horizontal shift
input ENUM_MA_METHOD InpMAMethod=MODE_SMMA;   // Smoothing method
```



Variable	Value
Period	13
Shift	0
Method	Smoothed

Note: Arrays and variables of complex types can't act as input variables.

Note: The length of a string comment for Input variables cannot exceed 63 characters.

Note: For input variables of string type, the limitation of the value length (string length) is set by the following conditions:

- the parameter value is represented by the "parameter_name=parameter_value" string ('=' is considered),
- maximum representation length of 255 characters (*total_length_max*=255 or 254 characters excluding '='),
- maximum length of the *parameter_name_length* string parameter = 63 characters.

Thus, the maximum string size for a string parameter is calculated using the equation:

```
parameter_value_length=total_length_max-parameter_name_length=254-parameter_name_lengt
```

This provides the maximum string size from 191 (parameter_name_length=63) to 253 characters (parameter_name_length=1).

Passing Parameters When Calling Custom Indicators from MQL5 Programs

Custom Indicators are called using the iCustom() function. After the name of the custom indicator, parameters should go in a strict accordance with the declaration of input variables of this custom indicator. If indicated parameters are less than input variables declared in the called custom indicator, the missing parameters are filled with values specified during the declaration of variables.

If the custom indicator uses the OnCalculate function of the first type (i.e., the indicator is calculated using the same array of data), then one of ENUM_APPLIED_PRICE values or handle of another indicator should be used as the last parameter when calling such a custom indicator. All parameters corresponding to input variables must be clearly indicated.

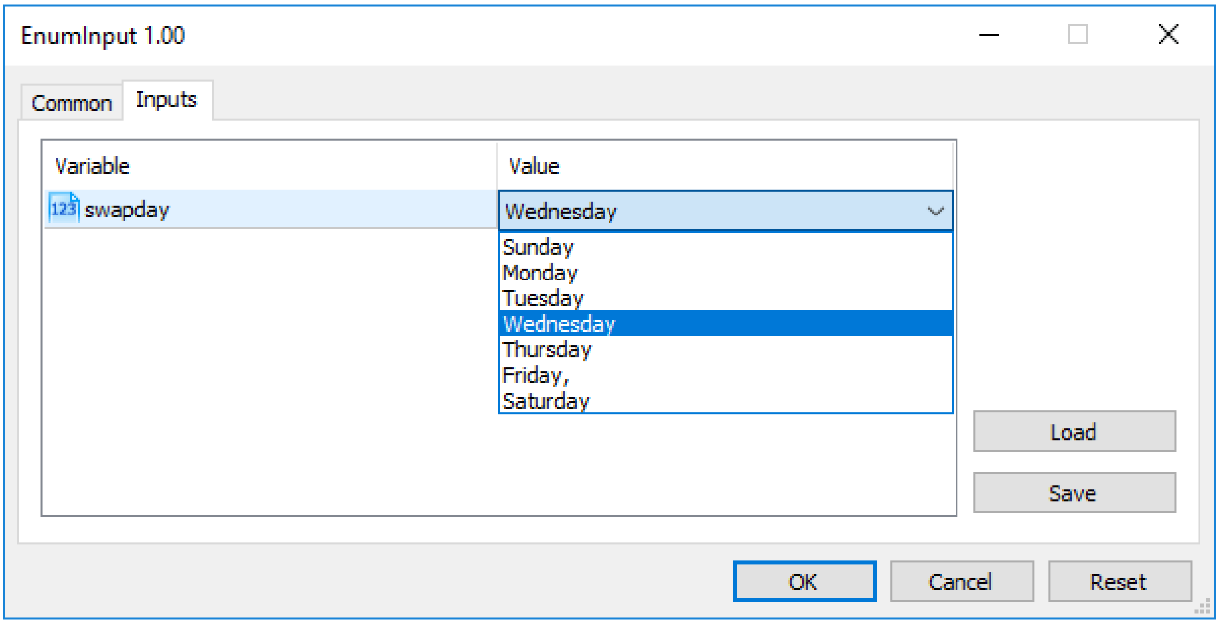
Enumerations as input Parameters

Not only built-in enumerations provided in MQL5, but also user defined variables can be used as input variables (input parameters for mql5 programs). For example, we can create the dayOfWeek enumeration, describing days of the week, and use the input variable to specify a particular day of the week, not as a number, but in a more common way.

Example:

```
#property script_show_inputs
//--- day of week
enum dayOfWeek
  {
   S=0,     // Sunday
   M=1,     // Monday
   T=2,     // Tuesday
   W=3,     // Wednesday
   Th=4,    // Thursday
   Fr=5,    // Friday,
   St=6,    // Saturday
  };
//--- input parameters
input dayOfWeek swapday=W;
```

In order to enable a user to select a necessary value from the properties window during the script startup, we use the preprocessor command #property script_show_inputs. We start the script and can choose one of values of the dayOfWeek enumeration from the list. We start the EnumInput script and go to the Inputs tab. By default, the value of swapday (day of triple swap charge) is Wednesday (W = 3), but we can specify any other value, and use this value to change the program operation.


EnumInput 1.00

Variable	Value
swapday	Wednesday

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday,
Saturday

Number of possible values of an enumeration is limited. In order to select an input value the drop-down list is used. Mnemonic names of enumeration members are used for values displayed in the list.

If a comment is associated with a mnemonic name, as shown in this example, the comment content is used instead of the mnemonic name.

Each value of the dayOfWeek enumeration has its value from 0 to 6, but in the list of parameters, comments specified for each value will be shown. This provides additional flexibility for writing programs with clear descriptions of input parameters.

Variables with sinput Modifier

Variables with input modifier allow not only setting external parameters values when launching programs but are also necessary when optimizing trading strategies in the Strategy Tester. Each input variable excluding the one of a string type can be used in optimization.

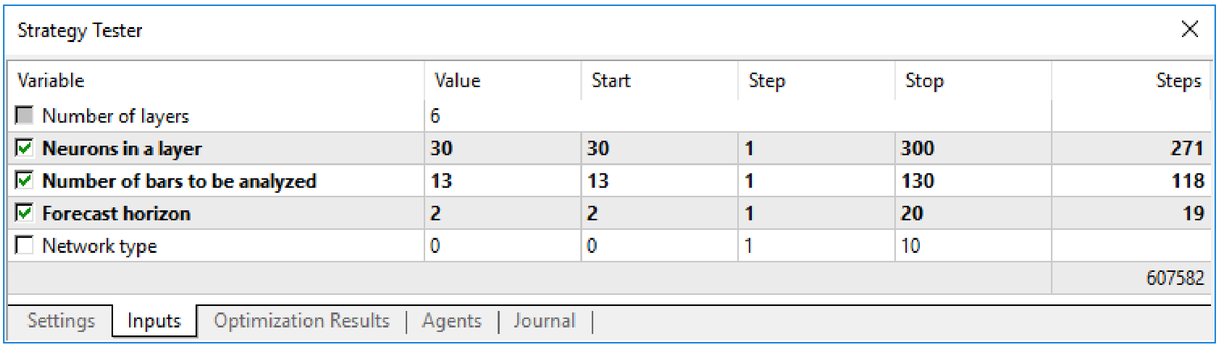
Sometimes, it is necessary to exclude some external program parameters from the area of all passes in the tester. sinput memory modifier has been introduced for such cases. sinput stands for static external variable declaration (sinput = static input). It means that the following declaration in an Expert Advisor code

```
sinput        int layers=6;   // Number of layers
```

will be equivalent to the full declaration

```
static input int layers=6;   // Number of layers
```

The variable declared with sinput modifier is an input parameter of MQL5 program. The value of this parameter can be changed when launching the program. However, this variable is not used in the optimization of input parameters. In other words, its values are not enumerated when searching for the best set of parameters fitting a specified condition.



Variable	Value	Start	Step	Stop	Steps
☐ Number of layers	6				
☑ **Neurons in a layer**	**30**	**30**	**1**	**300**	**271**
☑ **Number of bars to be analyzed**	**13**	**13**	**1**	**130**	**118**
☑ **Forecast horizon**	**2**	**2**	**1**	**20**	**19**
☐ Network type	0	0	1	10	
					607582

The Expert Advisor shown above has 5 external parameters. "Number of layers" is declared to be sinput and equal to 6. This parameter cannot be changed during a trading strategy optimization. We can specify the necessary value for it to be used further on. Start, Step and Stop fields are not available for such a variable.

Therefore, users will not be able to optimize this parameter after we specify sinput modifier for the variable. In other words, the terminal users will not be able to set initial and final values for it in the Strategy Tester for automatic enumeration in the specified range during optimization.

However, there is one exception to this rule: sinput variables can be varied in optimization tasks using ParameterSetRange() function. This function has been introduced specifically for the program control of available values sets for any input variable including the ones declared as static input (sinput). The ParameterGetRange() function allows to receive input variables values when optimization is launched (in OnTesterInit() handler) and to reset a change step value and a range, within which an optimized parameter values will be enumerated.

In this way, combining the sinput modifier and two functions that work with input parameters, allows to create a flexible rules for setting optimization intervals of input parameters that depend on values of another input parameters.

Arranging input parameters

For the convenience of working with MQL5 programs, the input parameters can be divided into named blocks using the group keyword. This allows for visual separation of some parameters from others based on the logic embedded in them.

```
input group             "Group name"
input int               variable1 = ...
input double            variable2 = ...
input double            variable3= ...
```

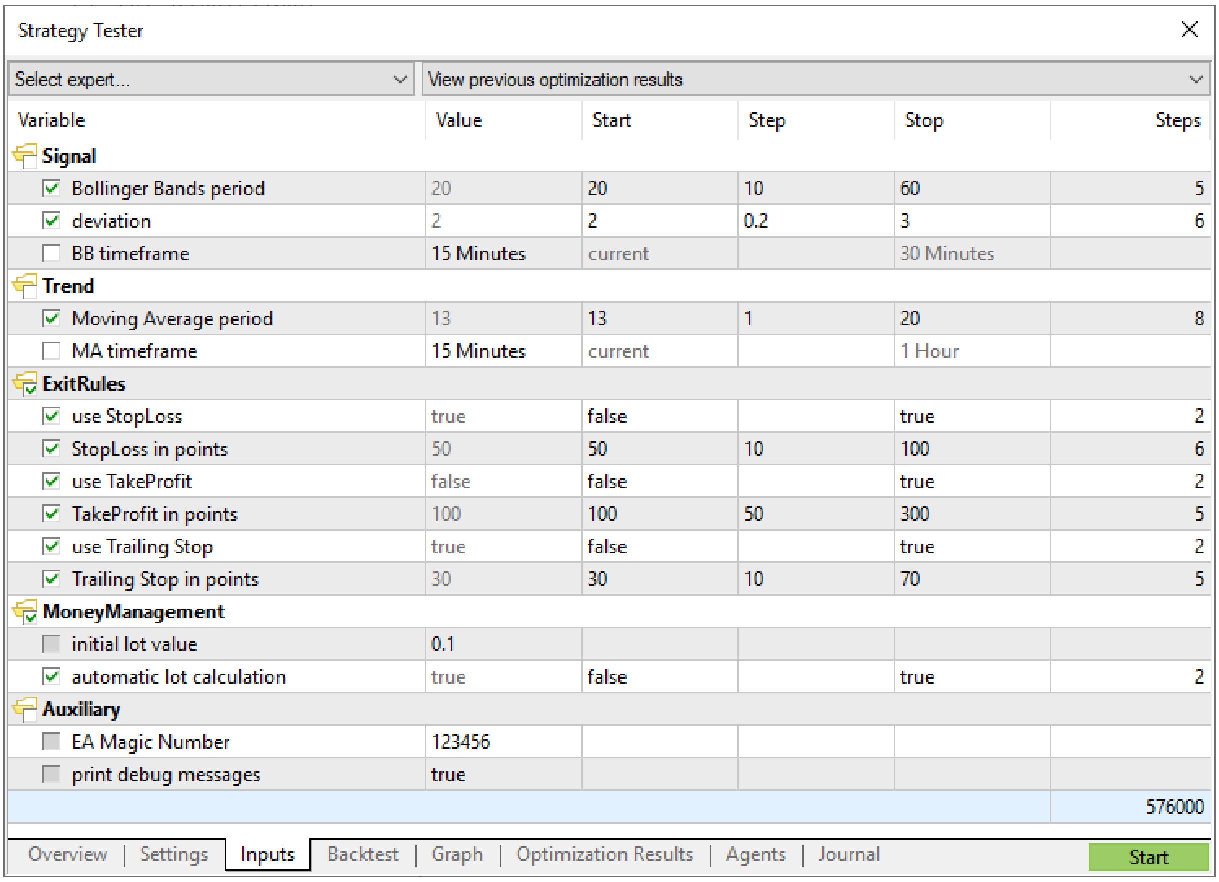
After such a declaration, all input parameters are visually joined into a specified group simplifying parameters configuration for MQL5 users when launching on a chart or in the strategy tester. Specification of each group is valid till a new group declaration appears:

```
input group             "Group name #1"
input int               group1_var1 = ...
input double            group1_var2 = ...
input double            group1_var3 = ...

input group             "Group name #2
input int               group2_var1 = ...
input double            group2_var2 = ...
input double            group2_var3 = ...
```

A sample EA featuring the block of inputs separated by their purpose:

```
input group             "Signal"
input int               ExtBBPeriod   = 20;         // Bollinger Bands period
input double            ExtBBDeviation= 2.0;        // deviation
input ENUM_TIMEFRAMES ExtSignalTF=PERIOD_M15;     // BB timeframe

input group             "Trend"
input int               ExtMAPeriod   = 13;         // Moving Average period
input ENUM_TIMEFRAMES ExtTrendTF=PERIOD_M15;      // MA timeframe

input group             "ExitRules"
input bool              ExtUseSL      = true;       // use StopLoss
input int               Ext_SL_Points = 50;         // StopLoss in points
input bool              ExtUseTP      = false;      // use TakeProfit
input int               Ext_TP_Points = 100;        // TakeProfit in points
input bool            ExtUseTS      = true;      // use Trailing Stop
input int             Ext_TS_Points = 30;        // Trailing Stop in points

input group           "MoneyManagement"
sinput double         ExtInitialLot = 0.1;       // initial lot value
input bool            ExtUseAutoLot = true;      // automatic lot calculation

input group           "Auxiliary"
sinput int            ExtMagicNumber = 123456;   // EA Magic Number
sinput bool           ExtDebugMessage= true;     // print debug messages
```

When launching such an EA in the strategy tester, you can double-click a group name to collapse/expand the inputs block, as well as click the group checkbox to select all its parameters for optimization.



Strategy Tester
Select expert...
View previous optimization results

Variable	Value	Start	Step	Stop	Steps
Signal					
☑ Bollinger Bands period	20	20	10	60	5
☑ deviation	2	2	0.2	3	6
☐ BB timeframe	15 Minutes	current		30 Minutes	
Trend					
☑ Moving Average period	13	13	1	20	8
☐ MA timeframe	15 Minutes	current		1 Hour	
ExitRules					
☑ use StopLoss	true	false		true	2
☑ StopLoss in points	50	50	10	100	6
☑ use TakeProfit	false	false		true	2
☑ TakeProfit in points	100	100	50	300	5
☑ use Trailing Stop	true	false		true	2
☑ Trailing Stop in points	30	30	10	70	5
MoneyManagement					
☐ initial lot value	0.1				
☑ automatic lot calculation	true	false		true	2
Auxiliary					
☐ EA Magic Number	123456				
☐ print debug messages	true				
					576000

Overview | Settings | Inputs | Backtest | Graph | Optimization Results | Agents | Journal
Start

See also

[iCustom], [Enumerations], [Properties of Programs]

Extern Variables

The extern keyword is used for declaring variable identifiers as identifiers of the static storage class with global lifetime. These variables exist from the start of the program and memory for them is allocated and initialized immediately after the start of the program.

You can create programs that consist of multiple source files; in this case a directive to the preprocessor #include is used. Variables declared as an extern with the same type and identifier can exist in different source files of one project.

When compiling the whole project, all the extern variables with the same type and an identifier are associated with one part of memory of the global variable pool. Extern variables are useful for separate compilation of source files. Extern variables can be initialized, but only once - existence of several initialized extern variables of the same type and with the same identifier is prohibited.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Initialization of Variables

Any variable can be initialized during definition. If a variable is not initialized explicitly, the value stored in this variable can be any. Implicit initialization is not used.

Global and static variables can be initialized only by a constant of the corresponding type or a constant expression. Local variables can be initialized by any expression, not just a constant.

Initialization of global and static variables is performed only once. Initialization of local variables is made every time you call the corresponding functions.

Examples:

```
int    n       = 1;
string s       = "hello";
double f[]     = { 0.0, 0.236, 0.382, 0.5, 0.618, 1.0 };
int    a[4][4] = { {1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {4, 4, 4, 4 } };
//--- from tetris
int    right[4]={WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER,
                 WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER};
//--- initialization of all fields of the structure with zero values
MqlTradeRequest request={};
```

List of values of the array elements must be enclosed in curly brackets. Missed initializing sequences are considered equal to 0.

If the size of the initialized array is not specified, it is determined by a compiler, based on the size of the initialization sequence.

Examples:

```
struct str3
  {
   int               low_part;
   int               high_part;
  };
struct str10
  {
   str3              s3;
   double            d1[10];
   int               i3;
  };
void OnStart()
  {
   str10 s10_1={{1,0},{1.0,2.1,3.2,4.4,5.3,6.1,7.8,8.7,9.2,10.0},100};
   str10 s10_2={{1,0},{},100};
   str10 s10_3={{1,0},{1.0}};
//---
   Print("1.  s10_1.d1[5] = ",s10_1.d1[5]);
   Print("2.  s10_2.d1[5] = ",s10_2.d1[5]);
   Print("3.  s10_3.d1[5] = ",s10_3.d1[5]);
   Print("4.  s10_3.d1[0] = ",s10_3.d1[0]);
  }
```

For structure type variable partial initialization is allowed, as well as for static arrays (with an implicitly set size). You can initialize one or more first elements of a structure or array, the other elements will be initialized with zeroes in this case.

See also

Data Types, Encapsulation and Extensibility of Types, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Visibility Scope and Lifetime of Variables

There are two basic types of scope: local scope and global scope.

A variable declared outside all functions is located into the global scope. Access to such variables can be done from anywhere in the program.These variables are located in the global pool of memory, so their lifetime coincides with the lifetime of the program.

A variable declared inside a block (part of code enclosed in curly brackets) belongs to the local scope. Such a variable is not visible (and therefore not available) outside the block, in which it is declared. The most common case of local declaration is a variable declared within a function. A variable declared locally, is located on the stack, and the lifetime of such a variable is equal to the lifetime of the function.

Since the scope of a local variable is the block in which it is declared, it is possible to declare variables with the same name, as those of variables declared in other blocks; as well as of those declared at upper levels, up to the global level.

Example:

```
void CalculateLWMA(int rates_total,int prev_calculated,int begin,const double &price[
  {
   int        i,limit;
   static int weightsum=0;
   double     sum=0;
//---
   if(prev_calculated==0)
     {
      limit=MA_Period+begin;
      //--- set empty value for first limit bars
      for(i=0; i<limit; i++) LineBuffer[i]=0.0;
      //--- calculate first visible value
      double firstValue=0;
      for(int i=begin; i<limit; i++)
        {
         int k=i-begin+1;
         weightsum+=k;
         firstValue+=k*price[i];
        }
      firstValue/=(double)weightsum;
      LineBuffer[limit-1]=firstValue;
     }
   else
     {
      limit=prev_calculated-1;
     }

   for(i=limit;i<rates_total;i++)
     {
      sum=0;
      for(int j=0; j<MA_Period; j++) sum+=(MA_Period-j)*price[i-j];
      LineBuffer[i]=sum/weightsum;
     }
//---
  }
```

Pay attention to the variable i, declared in line

```
      for(int i=begin; i<limit; i++)
        {
         int k=i-begin+1;
         weightsum+=k;
         firstValue+=k*price[i];
        }
```

Its scope is only the for loop; outside of this loop there is another variable with the same name, declared at the beginning of the function. In addition, the k variable is declared in the loop body, its scope is the loop body.

Local variables can be declared with the access specifier static. In this case, the compiler has a variable in the global pool of memory. Therefore, the lifetime of a static variable is equal to the lifetime of the program. Here the scope of such a variable is limited to the block in which it is declared.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Creating and Deleting Objects

Creating and Deleting Objects

After a MQL5 program is loaded for execution, memory is allocated to each variable according to its type. According to the access level, all variables are divided into two types - global variables and local variables. According to the memory class, they can be input parameters of a MQL5 program, static and automatic. If necessary, each variable is initialized by a corresponding value. After being used a variable is unintialized and memory used by it is returned to the MQL5 executable system.

Initialization and Deinitialization of Global Variables

Global variables are initialized automatically right after a MQL5 program is loaded and before any of function is called. During initialization initial values are assigned to variables of simple types and a constructor (if there is any) is called for objects. Input variables are always declared at a global level, and are initialized by values set by a user in the dialog during the program start.

Despite the fact that static variables are usually declared at a local level, the memory for these variables is pre-allocated, and initialization is performed right after a program is loaded, the same as for global variables.

The initialization order corresponds to the variable declaration order in the program. Deinitialization is performed in the reverse order. This rule is true only for the variables that were not created by the new operator. Such variables are created and initialized automatically right after loading, and are deinitialized before the program unloading.

Initialization and Deinitialization of Local Variables

If a variable declared on a local level is not a static one, memory is allocated automatically for such a variable. Local variables, as well as global ones, are initialized automatically at the moment when the program execution meets their declaration. Thus the initialization order corresponds to the order of declaration.

Local variables are deinitialized at the end of the program block, in which they were declared, and in the order opposite to their declaration. A program block is a compound operator that can be a part of selection operator switch, loop operator (for, while, do-while), a function body or a part of the if-else operator.

Local variables are initialized only at the moment when the program execution meets the variable declaration. If during the program execution the block, in which the variable is declared, was not executed, such a variable is not initialized.

Initialization and Deinitialization of Objects Placed

A special case is that with object pointers, because declaration of a pointer does not entail initialization of a corresponding objects. Dynamically placed objects are initialized only at the moment when the class sample is created by the new operator. Initialization of objects presupposes call of a constructor of a corresponding class. If there is no corresponding constructor in the class, its members of a simple type will not be automatically initialized; members of types string, dynamic array and complex object will be automatically initialized.

Pointers can be declared on a local or global level; and they can be initialized by the empty value of NULL or by the value of the pointer of the same or inherited type. If the *new* operator is called for a pointer declared on a local level, the *delete* operator for this pointer must be performed before exiting the level. Otherwise the pointer will be lost and the explicit deletion of the object will fail.

All objects created by the expression of *object_pointer*=new *Class_name*, must be then deleted by the delete(*object_pointer*) operator. If for some reasons such a variable is not deleted by the delete operator when the program is completed, the corresponding entry will appear in the "Experts" journal. One can declare several variables and assign a pointer of one object to all of them.

If a dynamically created object has a constructor, this constructor will be called at the moment of the *new* operator execution. If an object has a destructor, it will be called during the execution of the *delete* operator.

Thus dynamically placed objects are created only at the moment when the corresponding *new* operator is invoked, and are assuredly deleted either by the *delete* operator or automatically by the executing system of MQL5 during the program unloading. The order of declaration of pointers of dynamically created object doesn't influence the order of their initialization. The order of initialization and deinitialization is fully controlled by the programmer.

Dynamic memory allocation in MQL5

When working with dynamic arrays, released memory is immediately returned back to the operating system.

When working with dynamic class objects using the new operator, first memory is requested from the class memory pool the memory manager is working with. If there is not enough memory in the pool, memory is requested from the operating system. When deleting the dynamic object using the delete operator, released memory is immediately returned back to the class memory pool.

Memory manager releases memory back to the operating system immediately after exiting the following event handling functions: OnInit(), OnDeinit(), OnStart(), OnTick(), OnCalculate(), OnTimer(), OnTrade(), OnTester(), OnTesterInit(), OnTesterPass(), OnTesterDeinit(), OnChartEvent(), OnBookEvent().

Brief Characteristics of Variables

The main information about the order of creation, deletion, about calls of constructors and destructors is given in the below table.

	Global automatic variable	Local automatic variable	Dynamically created object
Initialization	right after a mql5 program is loaded	when the code line where it is declared is reached during execution	at the execution of the **new** operator
Initialization order	in the order of declaration	in the order of declaration	irrespective of the order of declaration
Deinitialization	before a mql5 program is unloaded	when execution exits the declaration block	when the **delete** operator is executed

	Global automatic variable	Local automatic variable	Dynamically created object
			or before a mql5 program is unloaded
Deinitialization order	in the order opposite to the initialization order	in the order opposite to the initialization order	irrespective of the initialization order
Constructor call	at mql5 program loading	at initialization	at the execution of the *new* operator
Destructor call	at mql5 program unloading	when exiting the block where the variable was initialized	at the execution of the *delete* operator
Error logs	log message in the "Experts" journal about the attempt to delete an automatically created object	log message in the "Experts" journal about the attempt to delete an automatically created object	log message in the "Experts" journal about undeleted dynamically created objects at the unload of a mql5 program

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables

Preprocessor

Preprocessor is a special subsystem of the MQL5 compiler that is intended for preparation of the program source code immediately before the program is compiled.

Preprocessor allows enhancement of the source code readability. The code can be structured by including of specific files containing source codes of mql5-programs. The possibility to assign mnemonic names to specific constants contributes to enhancement of the code readability.

Preprocessor also allows determining specific parameters of mql5-programs:

- Declare constants
- Set program properties
- Include files in program text
- Import functions
- Conditional Compilation

The preprocessor directives are used by the compiler to preprocess the source code before compiling it. The directive always begins with #, therefore the compiler prohibits using the symbol in names of variables, functions etc.

Each directive is described by a separate entry and is valid until the line break. You cannot use several directives in one entry. If the directive entry is too big, it can be broken into several lines using the '\' symbol. In this case, the next line is considered a continuation of the directive entry.

```
//+------------------------------------------------------------------+
//|  foreach pseudo-operator                                         |
//+------------------------------------------------------------------+
#define ForEach(index, array) for (int index = 0,                    \
   max_##index=ArraySize((array));                                   \
   index<max_##index; index++)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string array[]={"12","23","34","45"};
//--- bypass the array using ForEach
   ForEach(i,array)
     {
      PrintFormat("%d: array[%d]=%s",i,i,array[i]);
     }
  }
//+------------------------------------------------------------------+
/* Output result
   0: array[0]=12
   1: array[1]=23
   2: array[2]=34
   3: array[3]=45
*/
```

For the compiler, all these three #define directive lines look like a single long line. The example above also applies ## character which is a merge operator used in the #define macros to merge the two macro tokens into one. The tokens merge operator cannot be the first or last one in a macro definition.

Macro substitution (#define)

The preprocessor directives are used by the compiler to preprocess the source code before compiling it. The directive always begins with #, therefore the compiler prohibits using the symbol in names of variables, functions etc.

Each directive is described by a separate entry and is valid until the line break. You cannot use several directives in one entry. If the directive entry is too big, it can be broken into several lines using the '\' symbol. In this case, the next line is considered a continuation of the directive entry.

The #define directive can be used to assign mnemonic names to constants. There are two forms:

```
#define identifier expression                   // parameter-free form
#define identifier(par1,... par8) expression    // parametric form
```

The #define directive substitutes **expression** for all further found entries of *identifier* in the source text. The *identifier* is replaced only if it is a separate token. The *identifier* is not replaced if it is part of a comment, part of a string, or part of another longer identifier.

The constant identifier is governed by the same rules as variable names. The value can be of any type:

```
#define ABC               100
#define PI                3.14
#define COMPANY_NAME      "MetaQuotes Software Corp."
...
void ShowCopyright()
  {
   Print("Copyright  2001-2009, ",COMPANY_NAME);
   Print("https://www.metaquotes.net");
  }
```

expression can consist of several tokens, such as keywords, constants, constant and non-constant expressions. **expression** ends with the end of the line and can't be transferred to the next line.

Example:

```
#define TWO        2
#define THREE      3
#define INCOMPLETE TWO+THREE
#define COMPLETE   (TWO+THREE)
void OnStart()
  {
   Print("2 + 3*2 = ",INCOMPLETE*2);
   Print("(2 + 3)*2 = ",COMPLETE*2);
  }
// Result
// 2 + 3*2 = 8
// (2 + 3)*2 = 10
```

Parametric Form #define

With the parametric form, all the subsequent found entries of identifier will be replaced by expression taking into account the actual parameters. For example:

```
 // example with two parameters a and b
#define A 2+3
#define B 5-1
#define MUL(a, b) ((a)*(b))

double c=MUL(A,B);
Print("c=",c);
/*
expression double c=MUL(A,B);
is equivalent to double c=((2+3)*(5-1));
*/
// Result
// c=20
```

Be sure to enclose parameters in parentheses when using the parameters in expression, as this will help avoid non-obvious errors that are hard to find. If we rewrite the code without using the brackets, the result will be different:

```
 // example with two parameters a and b
#define A 2+3
#define B 5-1
#define MUL(a, b) a*b

double c=MUL(A,B);
Print("c=",c);
/*
expression double c=MUL(A,B);
is equivalent to double c=2+3*5-1;
*/
// Result
// c=16
```

When using the parametric form, maximum 8 parameters are allowed.

```
 // correct parametric form
#define LOG(text)  Print(__FILE__,"(",__LINE__,") :",text)   // one parameter - 'text

 // incorrect parametric form
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9)   p1+p2+p3+p4 // more than 8 pa
```

The #undef directive

The #undef directive cancels declaration of the macro substitution, defined before.

Example:

```
#define MACRO
```

```
void func1()
  {
#ifdef MACRO
   Print("MACRO is defined in ",__FUNCTION__);
#else
   Print("MACRO is not defined in ",__FUNCTION__);
#endif
  }

#undef MACRO

void func2()
  {
#ifdef MACRO
   Print("MACRO is defined in ",__FUNCTION__);
#else
   Print("MACRO is not defined in ",__FUNCTION__);
#endif
  }

void OnStart()
  {
   func1();
   func2();
  }

/* Result:
 MACRO is defined in func1
 MACRO is not defined in func2
*/
```

See also

Identifiers, Character Constants

Program Properties (#property)

Every mql5-program allows to specify additional specific parameters named #property that help client terminal in proper servicing for programs without the necessity to launch them explicitly. This concerns external settings of indicators, first of all. Properties described in included files are completely ignored. Properties must be specified in the main mq5-file.

```
#property identifier value
```

The compiler will write declared values in the configuration of the module executed.

Constant	Type	Description
icon	string	Path to the file of an image that will be used as an icon of the EX5 program. Path specification rules are the same as for resources. The property must be specified in the main module with the MQL5 source code. The icon file must be in the ICO format.
link	string	Link to the company website
copyright	string	The company name
version	string	Program version, maximum 31 characters
description	string	Brief text description of a mql5-program. Several *description* can be present, each of them describes one line of the text. The total length of all *description* can not exceed 511 characters including line feed.
stacksize	int	MQL5 program stack size. The stack of sufficient size is necessary when executing function recursive calls. When launching a script or an Expert Advisor on the chart, the stack of at least 8 MB is allocated. In case of indicators, the stack size is always fixed and equal to 1 MB. When a program is launched in the strategy tester, the stack of 16 MB is always allocated for it.
library		A library; no start function is assigned, functions with the export modifier can be imported in other mql5-programs
indicator_applied_price	int	Specifies the default value for the "Apply to" field. You can specify one of the values of ENUM_APPLIED_PRICE. If the property is not specified, the default value is PRICE_CLOSE
indicator_chart_window		Show the indicator in the chart window
indicator_separate_window		Show the indicator in a separate window

Constant	Type	Description
indicator_height	int	Fixed height of the indicator subwindow in pixels (property INDICATOR_HEIGHT)
indicator_buffers	int	Number of buffers for indicator calculation
indicator_plots	int	Number of graphic series in the indicator
indicator_minimum	double	The bottom scaling limit for a separate indicator window
indicator_maximum	double	The top scaling limit for a separate indicator window
indicator_labelN	string	Sets a label for the N-th graphic series displayed in DataWindow. For graphic series requiring multiple indicator buffers (DRAW_CANDLES, DRAW_FILLING and others), the label names are defined using the separator ';'.
indicator_colorN	color	The color for displaying line N, where N is the number of graphic series; numbering starts from 1
indicator_widthN	int	Line thickness in graphic series, where N is the number of graphic series; numbering starts from 1
indicator_styleN	int	Line style in graphic series, specified by the values of ENUM_LINE_STYLE. N is the number of graphic series; numbering starts from 1
indicator_typeN	int	Type of graphical plotting, specified by the values of ENUM_DRAW_TYPE. N is the number of graphic series; numbering starts from 1
indicator_levelN	double	Horizontal level of N in a separate indicator window
indicator_levelcolor	color	Color of horizontal levels of the indicator
indicator_levelwidth	int	Thickness of horizontal levels of the indicator
indicator_levelstyle	int	Style of horizontal levels of the indicator
script_show_confirm		Display a confirmation window before running the script
script_show_inputs		Display a window with the properties before running the script and disable this confirmation window
tester_indicator	string	Name of a custom indicator in the format of "*indicator_name.ex5*". Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a constant string. For all other cases (use of the IndicatorCreate() function or use of a non-constant string in the parameter

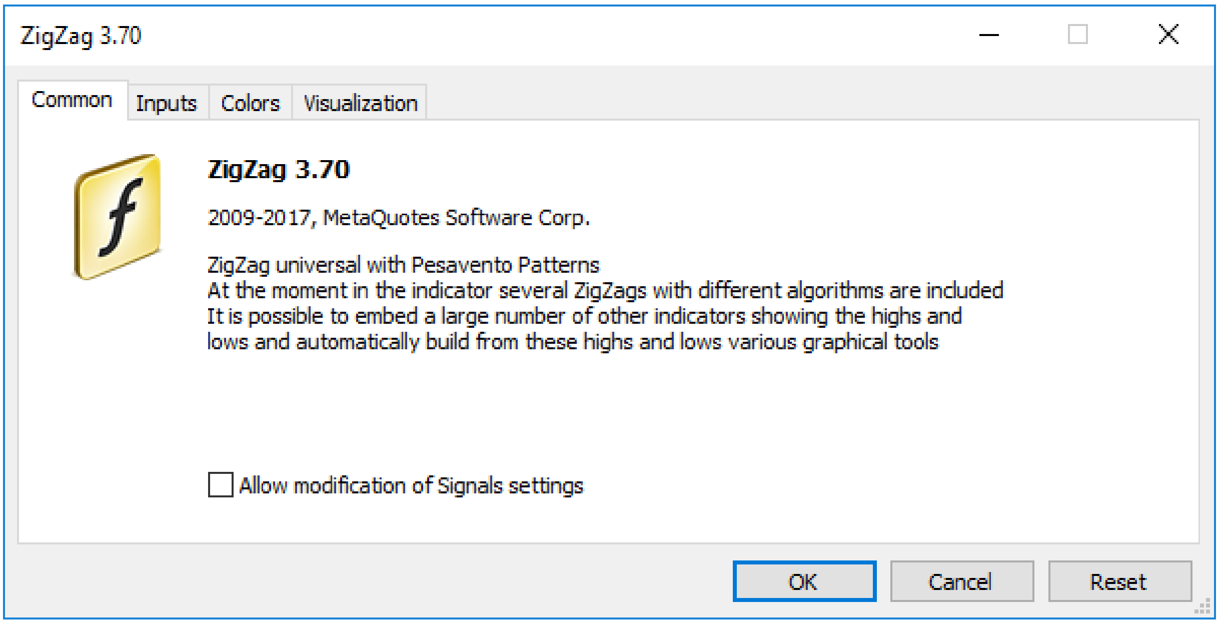
Constant	Type	Description
		that sets the indicator name) this property is required
tester_file	string	File name for a tester with the indication of extension, in double quotes (as a constant string). The specified file will be passed to tester. Input files to be tested, if there are necessary ones, must always be specified.
tester_library	string	Library name with the extension, in double quotes. A library can have 'dll' or 'ex5' as file extension. Libraries that require testing are defined automatically. However, if any of libraries is used by a custom indicator, this property is required
tester_set	string	Name of the set file with the values and the step of the input parameters. The file is passed to tester before testing and optimization. The file name is specified with an extension and double quotes as a constant string. If you specify the EA name and the version number as "<expert_name>_<number>.set" in a set file name, then it is automatically added to the parameter versions download menu under the <number> version number. For example, the name "MACD Sample_4.set" means that this is a set file for the "MACD Sample.mq5" EA with the version number equal to 4. To study the format, we recommend that you manually save the test/optimization settings in the strategy tester and then open the set file created in this way.
tester_no_cache	string	When performing optimization, the strategy tester saves all results of executed passes to the optimization cache, in which the test result is saved for each set of the input parameters. This allows using the ready-made results during re-optimization on the same parameters without wasting time on re-calculation. But in some tasks (for example, in math calculations), it may be necessary to carry out calculations regardless of the availability of ready-made results in the optimization cache. In this case, the file should include the *tester_no_cache* property. The test results are still stored in the

Constant	Type	Description
		cache, so that you can see all the data on performed passes in the strategy tester.
tester_everytick_calculate	string	In the Strategy Tester, indicators are only calculated when their data are accessed, i.e. when the values of indicator buffers are requested. This provides a significantly faster testing and optimization speed, if you do not need to obtain indicator values on each tick. By specifying the *tester_everytick_calculate* property, you can enable the forced calculation of the indicator on every tick. Indicators in the Strategy Tester are also forcibly calculated on every tick in the following cases: • when testing in the visual mode; • if the indicator has any of the following functions: EventChartCustom, OnChartEvent, OnTimer; • if the indicator was created using the compiler with build number below 1916. This feature only applies in the Strategy Tester, while in the terminal indicators are always calculated on each received tick.
optimization_chart_mode	string	Specifies the chart type and the names of two input parameters which will be used for the visualization of optimization results. For example, "3d, InpX, InpY" means that the results will be shown in a 3D chart with the coordinate axes based on the tested InpX and InpY parameter values. Thus, the property enables the specification of parameters that will be used to display the optimization chart and the chart type, directly in the program code. Possible options: • "3d, input_parameter_name1, input_parameter_name2" means a 3D visualization chart, which can be rotated, zoomed in and out. The chart is built using two parameters. • "2d, input_parameter_name1, input_parameter_name2" means a 2D grid chart, in which each cell is painted in a certain color depending on the result. The chart is built using two parameters.

Constant	Type	Description
		• "1d, input_parameter_name1, input_parameter_name2" means a linear chart, in which the results are sorted by the specified parameter. Each pass is displayed as a point. The chart is built based on one parameter. • "0d, input_parameter_name1, input_parameter_name2" means a regular chart with results sorted by the pass result arrival time. Each pass is displayed as a point in the chart. Parameter indication is not required, but the specified parameters can be used for the manual switch to other chart types. Optionally, you can indicate only the chart type, without specifying one or two input parameters. In this case, the terminal will select the required parameters to show the optimization chart.

Sample Task of Description and Version Number

```
#property version     "3.70"      // Current version of the Expert Advisor
#property description "ZigZag universal with Pesavento Patterns"
#property description "At the moment in the indicator several ZigZags with different a
#property description "It is possible to embed a large number of other indicators show
#property description "lows and automatically build from these highs and lows various
```



ZigZag 3.70

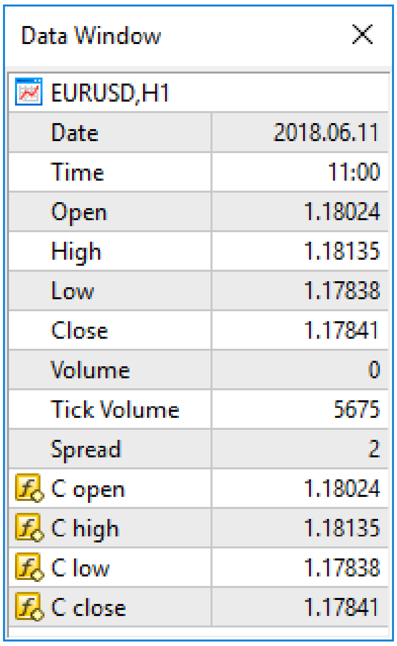
2009-2017, MetaQuotes Software Corp.

ZigZag universal with Pesavento Patterns
At the moment in the indicator several ZigZags with different algorithms are included
It is possible to embed a large number of other indicators showing the highs and
lows and automatically build from these highs and lows various graphical tools

☐ Allow modification of Signals settings

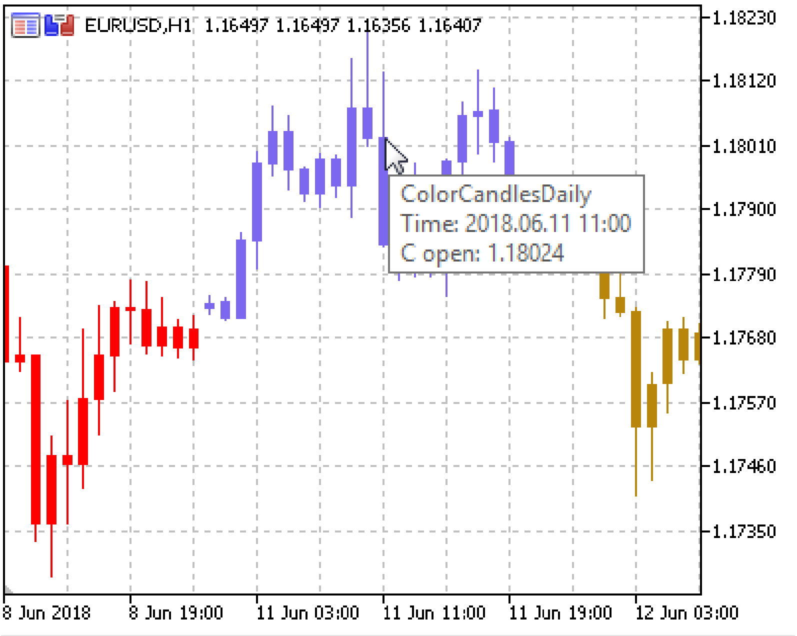
Examples of Specifying a Separate Label for Each Indicator Buffer ("C open; C high; C low; C close")

```
#property indicator_chart_window
#property indicator_buffers 4
#property indicator_plots   1
#property indicator_type1   DRAW_CANDLES
#property indicator_width1  3
#property indicator_label1  "C open;C high;C low;C close"
```



Data Window

EURUSD,H1	
Date	2018.06.11
Time	11:00
Open	1.18024
High	1.18135
Low	1.17838
Close	1.17841
Volume	0
Tick Volume	5675
Spread	2
C open	1.18024
C high	1.18135
C low	1.17838
C close	1.17841



EURUSD,H1 1.16497 1.16497 1.16356 1.16407

ColorCandlesDaily
Time: 2018.06.11 11:00
C open: 1.18024

Including Files (#include)

The *#include* command line can be placed anywhere in the program, but usually all inclusions are placed at the beginning of the source code. Call format:

```
#include <file_name>
#include "file_name"
```

Examples:

```
#include <WinUser32.mqh>
#include "mylib.mqh"
```

The preprocessor replaces the line *#include <file_name>* with the content of the file WinUser32.mqh. Angle brackets indicate that the WinUser32.mqh file will be taken from the standard directory (usually it is *terminal_installation_directory\MQL5\Include*). The current directory is not included in the search.

If the file name is enclosed in quotation marks, the search is made in the current directory (which contains the main source file). The standard directory is not included in the search.

See also

Standard Library, Importing Functions

Importing Function (#import)

Functions are imported from compiled MQL5 modules (*.ex5 files) and from operating system modules (*.dll files). The module name is specified in the *#import* directive. For compiler to be able to correctly form the imported function call and organize proper transmission parameters, the full description of functions is needed. Function descriptions immediately follow the *#import "module name"* directive. New command *#import* (can be without parameters) completes the block of imported function descriptions.

```
#import "file_name"
    func1 define;
    func2 define;
    ...
    funcN define;
#import
```

Imported functions can have any names. Functions having the same names but from different modules can be imported at the same time. Imported functions can have names that coincide with the names of built-in functions. Operation of scope resolution defines which of the functions should be called.

The order of searching for a file specified after the #import keyword is described in Call of Imported Functions.

Since the imported functions are outside the compiled module, the compiler can not verify the validity of passed parameters. Therefore, to avoid run-time errors, one must accurately describe the composition and order of parameters passed to imported functions. Parameters passed to imported functions (both from EX5, and from the DLL-module) can have default values.

The following can't be used for parameters in imported functions:

- pointers (*);
- links to objects that contain dynamic arrays and/or pointers.

Classes, string arrays or complex objects that contain strings and/or dynamic arrays of any types cannot be passed as a parameter to functions imported from DLL.

Examples:

```
#import "stdlib.ex5"
string ErrorDescription(int error_code);
int    RGB(int red_value,int green_value,int blue_value);
bool   CompareDoubles(double number1,double number2);
string DoubleToStrMorePrecision(double number,int precision);
string IntegerToHexString(int integer_number);
#import "ExpertSample.dll"
int    GetIntValue(int);
double GetDoubleValue(double);
string GetStringValue(string);
double GetArrayItemValue(double &arr[],int,int);
bool   SetArrayItemValue(double &arr[],int,int,double);
double GetRatesItemValue(double &rates[][6],int,int,int);
#import
```

To import functions during execution of a mql5 program, early binding is used. This means that the library is loaded during the loading of a program using its ex5 program.

It's not recommended to use a fully qualified name of the loadable module of type *Drive:\Directory\FileName.Ext*. MQL5 libraries are loaded from the *terminal_dir\MQL5\Libraries* folder.

If the imported function has different call versions for 32- and 64-bit Windows versions, both of them should be imported, and the right function version should be called explicitly using the _IsX64 variable.

Example:

```
#import "user32.dll"
//--- For the 32-bit system
int    MessageBoxW(uint hWnd,string lpText,string lpCaption,uint uType);
//--- For the 64-bit system
int    MessageBoxW(ulong hWnd,string lpText,string lpCaption,uint uType);
#import
//+------------------------------------------------------------------+
//|  MessageBox_32_64_bit uses the proper version of MessageBoxW()   |
//+------------------------------------------------------------------+
int MessageBox_32_64_bit()
  {
   int res=-1;
   //--- If we are using the 64-bit Windows
   if(_IsX64)
     {
      ulong hwnd=0;
      res=MessageBoxW(hwnd,"64-bit MessageBoxW call example","MessageBoxW 64 bit",MB_
     }
   else  // We are using the 32-bit Windows
     {
      uint hwnd=0;
      res=MessageBoxW(hwnd,"32-bit MessageBoxW call example","MessageBoxW 32 bit",MB_
     }
   return (res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int ans=MessageBox_32_64_bit();
   PrintFormat("MessageBox_32_64_bit returned %d",ans);
  }
```

Importing functions from .NET libraries

To work with .NET library functions, simply import DLL itself without defining specific functions. MetaEditor automatically imports all functions it is possible to work with:

- Simple structures (POD, plain old data) — structures that contain only simple data types.
- Public static functions having parameters, in which only simple types and POD structures or their arrays are used

To call functions from the library, simply import it:

```
#import "TestLib.dll"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int x=41;
   TestClass::Inc(x);
   Print(x);
  }
```

The C# code of the Inc function of the TestClass looks as follows:

```
public class TestClass
  {
   public static void Inc(ref int x)
     {
      x++;
     }
  }
```

As a result of execution, the script returns the value of 42.

See also

Including Files

Conditional Compilation (#ifdef, #ifndef, #else, #endif)

The preprocessor directives are used by the compiler to preprocess the source code before compiling it. The directive always begins with #, therefore the compiler prohibits using the symbol in names of variables, functions etc.

Each directive is described by a separate entry and is valid until the line break. You cannot use several directives in one entry. If the directive entry is too big, it can be broken into several lines using the '\' symbol. In this case, the next line is considered a continuation of the directive entry.

Preprocessor conditional compilation directives allow compiling or skipping a part of the program depending on the fulfillment of a certain condition.

That condition can take one of the following forms.

```
#ifdef identifier
   // the code located here is compiled if the identifier has already been defined fo
#endif
```

```
#ifndef identifier
   // the code located here is compiled if the identifier is not currently defined by
#endif
```

Any of the conditional compilation directives can be followed by any number of lines possibly containing #else directive and ending with #endif. If the verified condition is true, the lines between #else and #endif are ignored. If the verified condition is not fulfilled, all lines between checking and #else directive (or #endif directive if the former is absent) are ignored.

Example:

```
#ifndef TestMode
   #define TestMode
#endif
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   #ifdef TestMode
      Print("Test mode");
   #else
      Print("Normal mode");
   #endif
  }
```

Depending on the program type and compilation mode, the standard macros are defined the following way:

__MQL5__ macro is defined when compiling *.mq5 file, __MQL4__ macro is defined when compiling *.mq4 one.
_DEBUG macro is defined when compiling in debug mode.
_RELEASE macro is defined when compiling in release mode.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   #ifdef __MQL5__
      #ifdef _DEBUG
         Print("Hello from MQL5 compiler [DEBUG]");
      #else
        #ifdef _RELEASE
           Print("Hello from MQL5 compiler [RELEASE]");
        #endif
      #endif
   #else
      #ifdef __MQL4__
        #ifdef _DEBUG
           Print("Hello from MQL4 compiler [DEBUG]");
        #else
           #ifdef _RELEASE
              Print("Hello from MQL4 compiler [RELEASE]");
           #endif
        #endif
      #endif
   #endif
  }
```

Object-Oriented Programming

Object-oriented programming (OOP) is programming primarily focused on data, while data and behavior are being inseparably linked. Data and behavior together constitute a class, while objects are class instances.

The components of the object-oriented approach are:

- Encapsulation and type extensibility
- Inheritance
- Polymorphism
- Overloading
- Virtual functions

OOP considers computation as modeling of behavior. The modeled item is the object represented by computational abstractions. Suppose we want to write a well known game "Tetris". To do this, we must learn how to model the appearance of random shapes composed of four squares joined together by edges. Also we need to regulate the falling speed of shapes, define operations of rotation and shift of shapes. Moving of shapes on the screen is limited by the well's boundaries, this requirement must also be modeled. Besides that, filled rows of cubes must be destroyed and achieved points must be counted.

Thus, this easy-to-understand game requires the creation of several models - shape model, well model, shape movement model and so on. All these models are abstractions, represented by calculations in the computer. To describe these models, the concept of Abstract Data Type, ADT (or complex data type) is used. Strictly speaking, the model of the "shapes" motion in the DOM is not a data type, but it is a set of operations on the "shape" data type, using the restrictions of the "well" data type.

Objects are class variables. Object-oriented programming allows you to easily create and use ADT. Object-oriented programming uses the inheritance mechanism. The benefit of inheritance is in the fact that it allows obtaining derivative types from data types already defined by a user.

For example, to create Tetris shapes, it's convenient to create a base class Shape first. The other classes representing all seven possible shape types can be derived on its basis. Behavior of shapes is defined in the base class, while implementation of behavior of each separate shape is defined in derivative classes.

In OOP objects are responsible for their behavior. ADT developer should include a code to describe any behavior that would normally be expected from the corresponding objects. The fact that the object itself is responsible for its behavior, greatly simplifies the task of programming for the user of this object.

If we want to draw a shape on the screen, we need to know where the center will be and how to draw it. If a separate shape knows how to draw itself, the programmer should send a "draw" message when using such a shape.

The MQL5 Language is a C++ like, and it also has the encapsulation mechanism for the implementation of ADT. On the one hand encapsulation combines the internal details of the implementation of a particular type, and on the other hand it combines externally accessible functions that can influence objects of this type. Implementation details may be inaccessible for a program that uses this type.

The concept of OOP has a set of related concepts, including the following:

- Simulation of actions from the real world
- User-defined data types
- Hiding the implementation details
- Possibility of the code reuse through inheritance
- Interpretation of function calls during execution

Some of these concepts are rather vague, some are abstract, others are general.

Encapsulation and Extensibility of Types

OOP is a balanced approach to writing software. Data and behavior are packed together. This encapsulation creates user-defined data types, extending the language data types and interacting with them. Types extensibility is an opportunity to add to the language user-defined data types, which are also easy to use, as well as basic types.

An abstract data type, for example, a string, is a description of the ideal, well known behavior type.

The string user knows that the string operations, such as concatenation or print, have a certain behavior. Concatenation and print operations are called methods.

A certain implementation of ADT may have some restrictions, for example, strings can be limited in length. These limitations affect the behavior opened to all. At the same time, internal or private implementation details do not affect directly the way the user sees the object. For example, the string is often implemented as an array, while the internal base address of this array and its name are not essential for the user.

Encapsulation is the ability to hide the implementation details when the open interfaces to user-defined type is provided. In MQL5, as well as in C++, class and structure definitions (class and struct) are used for the encapsulation provisions in combination with access keywords private, protected and public.

The public keyword shows that access to the members that stand behind it is open without restrictions. Without this keyword, class members are locked by default. Private members are accessible only by member functions only of its class.

Protected class functions are available to class functions not only in its class, but also in its inheritor classes. Public class functions are available for any function within the scope of the class declaration. The protection makes possible to hide part of the class implementation, thus preventing unexpected changes in the structure of data. Access restriction or data hiding is a feature of the object-oriented programming.

Usually, class functions are protected and declared with the protected modifier, the reading and writing of the values are performed by using special so-called set-and get-methods that are defined by the public access modifier.

Example:

```
class CPerson
  {
protected:
   string            m_name;                     // name
public:
   void              SetName(string n){m_name=n;}// sets name
   string            GetName(){return (m_name);} // returns name
  };
```

This approach offers several advantages. First, by function name we can understand what it does - sets or gets the value of a class member. Secondly, perhaps in the future we will need to change the type of the m_name variable in the CPerson class or in any of its derivative classes.

In this case, we'll need just to change the implementation of functions SetName() and GetName(), while objects of the CPerson class will be available for using in a program without any code changes because the user will not even know that the data type of m_name has changed.

Example:

```
struct Name
  {
   string            first_name;                 // name
   string            last_name;                  // last name
  };

class CPerson
  {
protected:
   Name              m_name;                     // name
public:
   void              SetName(string n);
   string            GetName(){return(m_name.first_name+" "+m_name.last_name);}
private:
   string            GetFirstName(string full_name);
   string            GetLastName(string full_name);
  };

void CPerson::SetName(string n)
  {
   m_name.first_name=GetFirstName(n);
   m_name.last_name=GetLastName(n);
  }

string CPerson::GetFirstName(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>0) StringSetCharacter(full_name,pos,0);
   return(full_name);
  }

string CPerson::GetLastName(string full_name)
  {
   string ret_string;
   int pos=StringFind(full_name," ");
   if(pos>0) ret_string=StringSubstr(full_name,pos+1);
   else      ret_string=full_name;
   return(ret_string);
  }
```

See also

[Data Types]

Inheritance

The characteristic feature of OOP is the encouragement of code reuse through inheritance. A new class is made from the existing, which is called the base class. The derived class uses the members of the base class, but can also modify and supplement them.

Many types are variations of the existing types. It is often tedious to develop a new code for each of them. In addition, the new code implies new errors. The derived class inherits the description of the base class, thus any re-development and re-testing of code is unnecessary. The inheritance relationships are hierarchical.

Hierarchy is a method that allows to copy the elements in all their diversity and complexity. It introduces the objects classification. For example, the periodic table of elements has gases. They possess to properties inherent to all periodic elements.

Inert gases constitute the next important subclass. The hierarchy is that the inert gas, such as argon is a gas, and gas, in its turn, is part of the system. Such a hierarchy allows to interpret behaviour of inert gases easily. We know that their atoms contain protons and electrons, that is true for all other elements.

We know that they are in a gaseous state at room temperature, like all the gases. We know that no gas from inert gas subclass enters usual chemical reaction with other elements, and it is a property of all inert gases.

Consider an example of the inheritance of geometric shapes. To describe the whole variety of simple shapes (circle, triangle, rectangle, square etc.), the best way is to create a base class (ADT), which is the ancestor of all the derived classes.

Let's create a base class CShape, which contains just the most common members describing the shape. These members describe properties that are characteristic of any shape - the type of the shape and main anchor point coordinates.

Example:

```
//--- The base class Shape
class CShape
  {
protected:
   int       m_type;                   // Shape type
   int       m_xpos;                   // X - coordinate of the base point
   int       m_ypos;                   // Y - coordinate of the base point
public:
             CShape(){m_type=0; m_xpos=0; m_ypos=0;} // constructor
   void      SetXPos(int x){m_xpos=x;} // set X
   void      SetYPos(int y){m_ypos=y;} // set Y
  };
```

Next, create new classes derived from the base class, in which we will add necessary fields, each specifying a certain class. For the Circle shape it is necessary to add a member that contains the radius value. The Square shape is characterized by the side value. Therefore, derived classes, inherited from the base class CShape will be declared as follows:

```
//--- The derived class circle
class CCircle : public CShape        // After a colon we define the base class
  {                                    // from which inheritance is made
private:
   int             m_radius;           // circle radius

public:
                   CCircle(){m_type=1;}// constructor, type 1
  };
```

For the Square shape class declaration is similar:

```
//--- the derived class Square
class CSquare : public CShape        // After a colon we define the base class
  {                                    // from which inheritance is made
private:
   int            m_square_side;       // square side

public:
                  CSquare(){m_type=2;} // constructor, type 2
  };
```

It should be noted that while object is created the base class constructor is called first, and then the constructor of the derived class is called. When an object is destroyed first the destructor of the derived class is called, and then a base class destructor is called.

Thus, by declaring the most general members in the base class, we can add an additional members in derived classes, which specify a particular class. Inheritance allows creating powerful code libraries that can be reused many times.

The syntax for creating a derived class from an already existing one is as follows:

```
class class_name :
          (public | protected | private) opt  base_class_name
  {
    class members declaration
  };
```

One of aspects of the derived class is the visibility (openness) of its members successors (heirs). The public, protected and private keywords are used to indicate the extent, to which members of the base class will be available for the derived one. The public keyword after a colon in the header of a derived class indicates that the protected and public members of the base class CShape should be inherited as protected and public members of the derived class CCircle.

The private class members of the base class are not available for the derived class. The public inheritance also means that derived classes (CCircle and CSquare) are CShapes. That is, the Square (CSquare) is a shape (CShape), but the shape does not necessarily have to be a square.

The derived class is a modification of the base class, it inherits the protected and public members of the base class. The constructors and destructors of the base class cannot be inherited. In addition to members of the base class, new members are added in a derivative class.

The derived class may include the implementation of member functions, different from the base class. It has nothing common with an overload, when the meaning of the same function name may be different for different signatures.

In protected inheritance, public and protected members of base class become protected members of derived class. In private inheritance, the public and protected members of base class become private members of the derived class.

In protected and private inheritance, the relation that "the object of a derivative class is object of a base class" is not true. The protected and private inheritance types are rare, and each of them needs to be used carefully.

It should be understood that the type of inheritance (public, protected or private) does not affect the ways of **accessing the members of base classes in the hierarchy of inheritance from a derived class**. With any type of inheritance, only base class members declared with public and protected access specifiers will be available out of the derived classes. Let's consider it in the following example:

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Example class with a few access types                            |
//+------------------------------------------------------------------+
class CBaseClass
  {
private:             //--- The private member is not available from derived classes
   int               m_member;
protected:           //--- The protected method is available from the base class and
   int               Member(){return(m_member);}
public:              //--- Class constructor is available to all members of classes
                     CBaseClass(){m_member=5;return;};
private:             //--- A private method for assigning a value to m_member
   void              Member(int value) { m_member=value;};

  };
//+------------------------------------------------------------------+
//| Derived class with errors                                        |
//+------------------------------------------------------------------+
class CDerived: public CBaseClass // specification of public inheritance can be omitte
  {
public:
   void Func() // In the derived class, define a function with calls to base class me
     {
      //--- An attempt to modify a private member of the base class
      m_member=0;        // Error, the private member of the base class is not availa
      Member(0);         // Error, the private method of the base class is not availa
      //--- Reading the member of the base class
      Print(m_member);   // Error, the private member of the base class is not availab
      Print(Member());   // No error, protected method is available from the base clas
     }
  };
```

In the above example, CBaseClass has only a public method – the constructor. Constructors are called automatically when creating a class object. Therefore, the private member m_member and the protected methods Member() cannot be called from the outside. But in case of public inheritance, the Member() method of the base class will be available from the derived classes.

In case of protected inheritance, all the members of the base class with public and protected access become protected. It means that if public data members and methods of the base class were accessible from the outside, with protected inheritance they are available only from the classes of the derived class and its further derivatives.

```
//+------------------------------------------------------------------+
//| Example class with a few access types                            |
//+------------------------------------------------------------------+
class CBaseMathClass
  {
private:             //--- The private member is not available from derived classes
   double            m_Pi;
public:              //--- Getting and setting a value for m_Pi
   void              SetPI(double v){m_Pi=v;return;};
   double            GetPI(){return m_Pi;};
public:              // The class constructor is available to all members
                     CBaseMathClass() {SetPI(3.14);  PrintFormat("%s",__FUNCTION__);};
  };
//+------------------------------------------------------------------+
//| Derived class, in which m_Pi cannot be modified                  |
//+------------------------------------------------------------------+
class CProtectedChildClass: protected CBaseMathClass // Protected inheritance
  {
private:
   double            m_radius;
public:              //--- Public methods in the derived class
   void              SetRadius(double r){m_radius=r; return;};
   double            GetCircleLength(){return GetPI()*m_radius;};
  };
//+------------------------------------------------------------------+
//| Script starting function                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- When creating a derived class, the constructor of the base class will be called
   CProtectedChildClass pt;
//--- Specify radius
   pt.SetRadius(10);
   PrintFormat("Length=%G",pt.GetCircleLength());
//--- If we uncomment the line below, we will get an error at the stage of compilatior
// pt.SetPI(3);

//--- Now declare a variable of the base class and try to set the Pi constant equal tc
   CBaseMathClass bc;
   bc.SetPI(10);
//--- Here is the result
   PrintFormat("bc.GetPI()=%G",bc.GetPI());
  }
```

The example shows that methods SetPI() and GetPi() in the base class CBaseMathClass are open and available for calling from any place of the program. But at the same time, for CProtectedChildClass which is derived from it these methods can be called only from the methods of the CProtectedChildClass class or its derived classes.

In case of private inheritance, all the members of the basic class with the public and protected access become private, and calling them becomes impossible in further inheritance.

MQL5 has no multiple inheritance.

See also

[Structures and Classes](#)

Polymorphism

Polymorphism is an opportunity for different classes of objects, related through inheritance, to respond in various ways when calling the same function element. It helps to create a universal mechanism describing the behavior of not only the base class, but also descendant classes.

Let's continue to develop a base class CShape, and define a member function GetArea(), designed to calculate the area of a shape. In all the descendant classes, produced by inheritance from the base class, we redefine this function in accordance with rules of calculating the area of a particular shape.

For a square (class CSquare), the area is calculated through its sides, for a circle (class CCircle), area is expressed through its radius etc. We can create an array to store objects of CShape type, in which both objects of a base class and those of all descendant classes can be stored. Further we can call the same function for each element of the array.

Example:

```
//--- Base class
class CShape
  {
protected:
   int            m_type;                // Shape type
   int            m_xpos;                // X - coordinate of the base point
   int            m_ypos;                // Y - coordinate of the base point
public:
   void           CShape(){m_type=0;};   // constructor, type=0
   int            GetType(){return(m_type);};// returns type of the shape
virtual
   double         GetArea(){return (0); }// returns area of the shape
  };
```

Now, all of the derived classes have a member function getArea(), which returns a zero value. The implementation of this function in each descendant will vary.

```
//--- The derived class Circle
class CCircle : public CShape            // After a colon we define the base class
  {                                      // from which inheritance is made
private:
   double         m_radius;              // circle radius

public:
   void           CCircle(){m_type=1;};  // constructor, type=1
   void           SetRadius(double r){m_radius=r;};
   virtual double GetArea(){return (3.14*m_radius*m_radius);}// circle area
  };
```

For the class Square the declaration is the same:

```
//--- The derived class Square
class CSquare : public CShape            // After a colon we define the base class
  {                                      // from which inheritance is made

private:
   double            m_square_side;        // square side

public:
   void              CSquare(){m_type=2;}; // constructor, type=1
   void              SetSide(double s){m_square_side=s;};
   virtual double    GetArea(){return (m_square_side*m_square_side);}// square area
  };
```

For calculating the area of the square and circle, we need the corresponding values of m_radius and m_square_side, so we have added the functions SetRadius() and SetSide() in the declaration of the corresponding class.

It is assumed that object of different types (CCircle and CSquare) derived from one base type CShape are used in our program. Polymorphism allows creating an array of objects of the base CShape class, but when declaring this array, these objects are yet unknown and their type is undefined.

The decision on what type of object will be contained in each element of the array will be taken directly during program execution. This involves the dynamic creation of objects of the appropriate classes, and hence the necessity to use object pointers instead of objects.

The new operator is used for dynamic creation of objects. Each such object must be individually and explicitly deleted using the delete operator. Therefore we will declare an array of pointers of CShape type, and create an object of a proper type for each element (**new Class_Name**), as shown in the following script example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare an array of object pointers of the base type
   CShape *shapes[5];   // An array of pointers to CShape object

//--- Here fill in the array with derived objects
//--- Declare a pointer to the object of CCircle type
   CCircle *circle=new CCircle();
//--- Set object properties at the circle pointer
   circle.SetRadius(2.5);
//--- Place the pointer value in shapes[0]
   shapes[0]=circle;

//--- Create another CCircle object and write down its pointer in shapes[1]
   circle=new CCircle();
   shapes[1]=circle;
   circle.SetRadius(5);

//--- Here we intentionally "forget" to set a value for shapes[2]
//circle=new CCircle();
//circle.SetRadius(10);
//shapes[2]=circle;
//--- Set NULL for the element that is not used
   shapes[2]=NULL;

//--- Create a CSquare object and write down its pointer to shapes[3]
   CSquare *square=new CSquare();
   square.SetSide(5);
   shapes[3]=square;

//--- Create a CSquare object and write down its pointer to shapes[4]
   square=new CSquare();
   square.SetSide(10);
   shapes[4]=square;

//--- We have an array of pointers, get its size
   int total=ArraySize(shapes);
//--- Pass in a loop through all pointers in the array
   for(int i=0; i<5;i++)
     {
      //--- If the pointer at the specified index is valid
      if(CheckPointer(shapes[i])!=POINTER_INVALID)
        {
         //--- Log the type and square of the shape
         PrintFormat("The object of type %d has the square %G",
               shapes[i].GetType(),
               shapes[i].GetArea());
        }
      //--- If the pointer has type POINTER_INVALID
      else
        {
         //--- Notify of an error
         PrintFormat("Object shapes[%d] has not been initialized! Its pointer is %s",
                     i,EnumToString(CheckPointer(shapes[i])));
        }
     }

//--- We must delete all created dynamic objects
   for(int i=0;i<total;i++)
     {
      //--- We can delete only the objects with pointers of POINTER_DYNAMIC type
      if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
        {
         //--- Notify of deletion
         PrintFormat("Deleting shapes[%d]",i);
         //--- Delete an object by its pointer
         delete shapes[i];
        }
     }
  }
```

Please note that when deleting an object using the delete operator, the type of its pointer must be checked. Only objects with the POINTER_DYNAMIC pointer can be deleted using delete. For pointers of other type, an error will be returned.

But besides the redefining of functions during inheritance, polymorphism also includes the implementation of one and the same functions with different sets of parameters within a class. This means that the class may have several functions with the same name but with a different type and/or set of parameters. In this case, polymorphism is implemented through the function overload.

See also

Standard Library

Overload

Within one class it is possible to define two or more methods that use the same name, but have different numbers of parameters. When this occurs, methods are called overloaded and such a process is referred to as method overloading.

Method overloading is one of ways of polymorphism realization. Overloading of methods is performed according to the same rules as the function overloading.

If the called function has no exact match, the compiler searches for a suitable function on three levels sequentially:

1. search within class methods.
2. search within the base class methods, consistently from the nearest ancestor to the very first.
3. search among other functions.

If there is no exact correspondence at all levels, but several suitable functions at different levels have been found, the function found at the least level is used. Within one level, there can't be more than one suitable function.

See also

Function Overloading

Virtual Functions

The virtual keyword is the function specifier, which provides a mechanism to select dynamically at runtime an appropriate function-member among the functions of basic and derived classes. Structures cannot have virtual functions. It can be used to change the declarations for function-members only.

The virtual function, like an ordinary function, must have an executable body. When called, its semantic is the same as that of other functions.

A virtual function may be overridden in a derived class. The choice of what function definition should be called for a virtual function is made dynamically (at runtime). A typical case is when a base class contains a virtual function, and derived classes have their own versions of this function.

The pointer to the base class can indicate either a base class object or the object of a derived class. The choice of the member-function to call will be performed at runtime and will depend on the type of the object, not the type of the pointer. If there is no member of a derived type, the virtual function of the base class is used by default.

Destructors are always virtual, regardless of whether they are declared with the virtual keyword or not.

Let's consider the use of virtual functions on the example of MT5_Tetris.mq5. The base class CTetrisShape with the virtual function Draw is defined in the included file MT5_TetrisShape.mqh.

```
//+------------------------------------------------------------------+
class CTetrisShape
  {
protected:
   int               m_type;
   int               m_xpos;
   int               m_ypos;
   int               m_xsize;
   int               m_ysize;
   int               m_prev_turn;
   int               m_turn;
   int               m_right_border;
public:
   void              CTetrisShape();
   void              SetRightBorder(int border) { m_right_border=border; }
   void              SetYPos(int ypos)          { m_ypos=ypos;            }
   void              SetXPos(int xpos)          { m_xpos=xpos;            }
   int               GetYPos()                  { return(m_ypos);         }
   int               GetXPos()                  { return(m_xpos);         }
   int               GetYSize()                 { return(m_ysize);        }
   int               GetXSize()                 { return(m_xsize);        }
   int               GetType()                  { return(m_type);         }
   void              Left()                     { m_xpos-=SHAPE_SIZE;     }
   void              Right()                    { m_xpos+=SHAPE_SIZE;     }
   void              Rotate()                   { m_prev_turn=m_turn; if(++m_turn>3) r
   virtual void      Draw()                     { return;                 }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

Further, for each derived class, this function is implemented in accordance with characteristics of a descendant class. For example, the first shape CTetrisShape1 has its own implementation of the Draw() function:

```
class CTetrisShape1 : public CTetrisShape
  {
public:
   //--- shape drawing
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      if(m_turn==0 || m_turn==2)
        {
         //--- horizontal
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
           }
        }
      else
        {
         //--- vertical
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos);
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+i*SHAPE_SIZE);
           }
        }
     }
  }
```

The Square shape is described by class CTetrisShape6 and has its own implementation of the Draw() method:

```
class CTetrisShape6 : public CTetrisShape
  {
public:
   //--- Shape drawing
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      for(i=0; i<2; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
        }
      for(i=2; i<4; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+(i-2)*SHAPE_SIZE);
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+SHAPE_SIZE);
        }
     }
  };
```

Depending on the class, to which the created object belongs, it calls the virtual function of this or that derived class.

```
void CTetrisField::NewShape()
  {
//--- creating one of the 7 possible shapes randomly
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- draw
   m_shape.Draw();
//---
  }
```

Modifier 'override'

The 'override' modifier means that the declared function must override the method of a parent class. Use of this method allows you to avoid overriding errors, for example it allows you to avoid accidental modification of the method signature. Suppose, the 'func' method is defined in the base class. The method accepts an int variable as an argument:

```
class CFoo
  {
   void virtual func(int x) const { }
  };
```

Next, the method is overridden in the child class:

```
class CBar : public CFoo
  {
   void func(short x) { }
  };
```

However, the argument type is mistakenly changed from int to short. In fact, this is not method overriding, but it is method overloading. Acting in accordance with the overloaded function defining algorithm, the compiler can in certain situations choose a method defined in the base class instead of the overridden method.

In order to avoid such errors, you should explicitly add the 'override' modifier to the method you want to override.

```
class CBar : public CFoo
  {
   void func(short x) override { }
  };
```

If the method signature is changed during overriding, the compiler will not be able to find a method with the same signature in the parent class, and it will return a compilation error:

```
'CBar::func' method is declared with 'override' specifier but does not override any ba
```

Modifier 'final'

The 'final' modifier does the opposite — it prohibits method overriding in child classes. If a method implementation is sufficient and fully complete, declare this method with the 'final' modifier so as to make sure that it will not be modified later.

```
class CFoo
  {
   void virtual func(int x) final { }
  };

class CBar : public CFoo
  {
   void func(int) { }
  };
```

If you try to override a method with the 'final' modifier as shown in the above example, the compiler will return an error:

```
'CFoo::func' method declared as 'final' cannot be overridden by 'CBar::func'
see declaration of 'CFoo::func'
```

See also

Standard Library

Static members of a Class/Structure

Static Members

The members of a class can be declared using the storage class modifier static. These data members are shared by all instances of this class and are stored in one place. Non-static data members are created for each class object variable.

The inability to declare static members of a class would have led to the need to declare these data on the the global level of the program. It would break the relationship between the data and their class, and is not consistent with the basic paradigm of the OOP - joining data and methods for handling them in a class. The static member allows class data that are not specific to a particular instance to exist in the class scope.

Since a static class member does not depend on the particular instance, the reference to it is as follows:

```
class_name::variable
```

where *class_name* is the name of the class, and *variable* is the name of the class member.

As you see, to access the static member of a class, context resolution operator :: is used. When you access a static member within class methods, the context operator is optional.

Static member of a class has to be explicitly initialized with desired value. For this it must be declared and initialized in global scope. The sequence of static members initialization will correspond to the sequence of their declaration in global scope.

For example, we have a class *CParser* used for parsing the text, and we need to count the total number of processed words and characters. We only need to declare the necessary class members as static and initialize them at the global level. Then all instances of the class will use common counters of words and characters.

```
//+------------------------------------------------------------------+
//| Class "Text analyzer"                                            |
//+------------------------------------------------------------------+
class CParser
  {
public:
   static int        s_words;
   static int        s_symbols;
   //--- Constructor and destructor
                     CParser(void);
                    ~CParser(void){};
  };
...
//--- Initialization of static members of the Parser class at the global level
int CParser::s_words=0;
int CParser::s_symbols=0;
```

A static class member can be declared with the *const* keyword. Such static constants must be initialized at the global level with the *const* keyword:

```
//+------------------------------------------------------------------+
//| Class "Stack" for storing processed data                         |
//+------------------------------------------------------------------+
class CStack
  {
public:
                      CStack(void);
                     ~CStack(void){};
...
private:
   static const int  s_max_length; // Maximum stack capacity
  };

//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
```

Pointer this

The keyword this denotes an implicitly declared pointer to itself – to a specific instance of the class, in the context of which the method is executed. It can be used only in non-static methods of the class. Pointer this is an implicit non-static member of any class.

In static functions you can access only static members/methods of a class.

Static Methods

In MQL5 member functions of type static can be used. The *static* modifier must precede the return type of a function in the declaration inside a class.

```
class CStack
  {
public:
   //--- Constructor and destructor
                      CStack(void){};
                     ~CStack(void){};
   //--- Maximum stack capacity
   static int         Capacity();
private:
   int                m_length;     // The number of elements in the stack
   static const int  s_max_length; // Maximum stack capacity
  };
//+------------------------------------------------------------------+
//| Returns the maximum number of elements to store in the stack     |
//+------------------------------------------------------------------+
int CStack::Capacity(void)
  {
   return(s_max_length);
  }
//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare CStack type variable
   CStack stack;
//--- call the object's static method
   Print("CStack.s_max_length=",stack.Capacity());
//--- it can also be called the following way, as the method is static and does not re
   Print("CStack.s_max_length=",CStack::Capacity());
  }
```

A method with the const modifier is called constant and cannot modify implicit members of its class. Declaration of constant functions of a class and constant parameters is called *const-correctness* control. Through this control you can be sure that the compiler will ensure the consistency of values of objects and will return an error during compilation if there is something wrong.

The const modifier is placed after the list of arguments inside a class declaration. Definition outside a class should also include the *const* modifier:

```
//+------------------------------------------------------------------+
//| Class "Rectangle"                                                |
//+------------------------------------------------------------------+
class CRectangle
  {
private:
   double            m_width;      // Width
   double            m_height;     // Height
public:
   //--- Constructors and destructor
                     CRectangle(void):m_width(0),m_height(0){};
                     CRectangle(const double w,const double h):m_width(w),m_height(h)
                    ~CRectangle(void){};
   //--- Calculating the area
   double            Square(void) const;
   static double     Square(const double w,const double h);// { return(w*h); }
  };
//+------------------------------------------------------------------+
//| Returns the area of the "Rectangle" object                       |
//+------------------------------------------------------------------+
double CRectangle::Square(void) const
  {
   return(Square(m_width,m_height));
  }
//+------------------------------------------------------------------+
//| Returns the product of two variables                             |
//+------------------------------------------------------------------+
static double CRectangle::Square(const double w,const double h)
  {
    return(w*h);
   }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Create a rectangle rect with the sides equal to 5 and 6
   CRectangle rect(5,6);
//--- Find the rectangle area using a constant method
   PrintFormat("rect.Square()=%.2f",rect.Square());
//--- Find the product of numbers using the static method of class CRectangle
   PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5));
  }
```

An additional argument in favor of using the constancy control is the fact that in this case, the compiler generates a special optimization, for example, places a constant object in read-only memory.

A static function cannot be determined with the const modifier, because this modifier ensures the constancy of the instance members when calling this function. But, as mentioned above, the static function cannot access non-static class members.

See also

Static Variables, Variables, References. Modifier & and Keyword this

Function templates

Overloaded functions are commonly used to perform similar operations on various data types. ArraySize() is a simple example of such function in MQL5. It returns size of any type of array. In fact, this system function is overloaded and the entire implementation of such an overload is hidden from MQL5 application developers:

```
int  ArraySize(
   void&  array[]      // checked array
   );
```

It means that MQL5 language compiler inserts necessary implementation for each call of this function. For example, that is how it can be done for integer type arrays:

```
int  ArraySize(
   int&  array[]      // array with int type elements
   );
```

ArraySize() function can be displayed the following way for MqlRates type array for working with quotations in historical data format:

```
int  ArraySize(
   MqlRates&  array[] // array filled with MqlRates type values
   );
```

Thus, it is very convenient to use the same function for working with different types. However, all preliminary work should be carried out – the necessary function should be overloaded for all data types it should correctly work with.

There is a convenient solution. If similar operations should be executed for each data type, it is possible to use function templates. In this case, a programmer needs to write only one function template description. When describing the template in such a way, we should specify only some formal parameter instead of some definite data type the function should work with. The compiler will automatically generate various functions for the appropriate handling of each type based on the types of the arguments used when calling the function.

Function template definition starts with the template keyword followed by the list of formal parameters in angle brackets. Each formal parameter is preceded by the typename keyword. Formal parameter types are built-in or user-defined types. They are used:

- to specify the types of function arguments,
- to specify the types of function's return value,
- to declare the variables inside the function definition

Number of template parameters cannot exceed eight. Each formal parameter in the template definition should appear in the list of function parameters at least once. Each name of the formal parameter should be unique.

Below is an example of a function template for searching the highest value in the array of any numeric type (integer and real numbers):

```
template<typename T>
T ArrayMax(T &arr[])
  {
   uint size=ArraySize(arr);
   if(size==0) return(0);

   T max=arr[0];
   for(uint n=1;n<size;n++)
      if(max<arr[n]) max=arr[n];
//---
   return(max);
  }
```

This template defines the function that finds the highest value in the passed array and returns this value as a result. Keep in mind that the ArrayMaximum() function built in MQL5 returns only the highest value index that can be used to find the value itself. For example:

```
//--- create an array
   double array[];
   int size=50;
   ArrayResize(array,size);
//---  fill with random values
   for(int i=0;i<size;i++)
     {
      array[i]=MathRand();
     }

//--- find position of the highest value in the array
   int max_position=ArrayMaximum(array);
//--- now, get the highest value itself in the array
   double max=array[max_position];
//--- display the found value
   Print("Max value = ",max);
```

Thus, we have performed two steps to get the highest value in the array. With ArrayMax() function template, we can get the result of the necessary type just by passing the array of an appropriate type into this function. It means that instead of two last lines

```
//--- find position of the highest value in the array
   int max_position=ArrayMaximum(array);
//--- now, get the highest value itself in the array
   double max=array[max_position];
```

we now can use only one line, in which the returned result has the same type as the array passed into function:

```
//--- find the highest value
   double max=ArrayMax(array);
```

In this case, the type of result returned by the ArrayMax() function will automatically match the type of array.

Use the typename keyword to get the argument type as a string in order to create general purpose methods of working with various data types. Let's consider a specific example of the function that returns data type as a string:

```
#include <Trade\Trade.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CTrade trade;
   double d_value=M_PI;
   int i_value=INT_MAX;
   Print("d_value: type=",GetTypeName(d_value), ",   value=", d_value);
   Print("i_value: type=",GetTypeName(i_value), ",   value=", i_value);
   Print("trade: type=",GetTypeName(trade));
//---
  }
//+------------------------------------------------------------------+
//| Type is returned as a line                                       |
//+------------------------------------------------------------------+
template<typename T>
string GetTypeName(const T &t)
  {
//--- return the type as a line
   return(typename(T));
//---
  }
```

Function templates can also be used for class methods, for example:

```
class CFile
  {
   ...
public:
   ...
   template<typename T>
   uint WriteStruct(T &data);
  };

template<typename T>
uint CFile::WriteStruct(T &data)
  {
   ...
   return(FileWriteStruct(m_handle,data));
  }
```

Function templates should not be declared with export, virtual and #import keywords.

Template function overload

A template function overload may be necessary sometimes. For example, we have a template function that writes the value of the second parameter to the first one using typecasting. MQL5 does not allow typecasting string to bool. We can do that ourselves – let's create an overload of a template function. For example:

```
//+------------------------------------------------------------------+
//| Template function                                                |
//+------------------------------------------------------------------+
template<typename T1,typename T2>
string Assign(T1 &var1,T2 var2)
  {
   var1=(T1)var2;
   return(__FUNCSIG__);
  }
//+------------------------------------------------------------------+
//| Special overload for bool+string                      |
//+------------------------------------------------------------------+
string Assign(bool &var1,string var2)
  {
   var1=(StringCompare(var2,"true",false) || StringToInteger(var2)!=0);
   return(__FUNCSIG__);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int i;
   bool b;
   Print(Assign(i,"test"));
   Print(Assign(b,"test"));
  }
```

As a result of the code execution, we can see that the Assign() template function has been used for the int+string pair, while the overloaded version has already been used for the bool+string pair during the second call.

```
string Assign<int,string>(int&,string)
string Assign(bool&,string)
```

See also

Overload

Template advantages

Function templates are used when you need to perform similar operations on various data types, for example, searching for a maximum element in the array. The main advantage of applying the templates is that you do not have to code a separate overload for each type. Instead of declaring multiple overloads of each type

```
double ArrayMax(double array[])
  {
   ...
  }
int ArrayMax(int array[])
  {
   ...
  }
uint ArrayMax(uint array[])
  {
   ...
  }
long ArrayMax(long array[])
  {
   ...
  }
datetime ArrayMax(datetime array[])
  {
   ...
  }
```

we need to write only one template function

```
template<typename T>
T ArrayMax(T array[])
  {
   if(ArraySize()==0)
      return(0);
   uint max_index=ArrayMaximum(array);
   return(array[max_index]);
  }
```

to use it in your code:

```
double high[];
datetime time[];
....
double max_high=ArrayMax(high);
datetime lasttime=ArrayMax(time);
```

Here, the *T* formal parameter specifying a type of used data is replaced with an actually applied type during compilation, i.e. the compiler automatically generates a separate function for each type - double, datetime, etc. MQL5 also allows you to develop class templates using all the advantages of the approach.

Class templates

A class template is declared using the template keyword followed by angle brackets<> enumerating the list of formal parameters with the typename keyword. This entry informs the compiler that it deals with a generic class with the *T* formal parameter defining a real variable type when implementing a class. For example, let's create a vector class for storing an array with *T* type elements:

```
#define TOSTR(x) #x+" "   // macro for displaying an object name
//+------------------------------------------------------------------+
//| Vector class for storing T-type elements                         |
//+------------------------------------------------------------------+
template <typename T>
class TArray
  {
protected:
   T                  m_array[];
public:
   //--- constructor creates an array for 10 elements by default
   void TArray(void){ArrayResize(m_array,10);}
   //--- constructor for creating a vector with a specified array size
   void TArray(int size){ArrayResize(m_array,size);}
   //--- return a type and amount of data stored in the TArray type object
   string Type(void){return(typename(m_array[0])+":"+(string)ArraySize(m_array));};
  };
```

Next, let's apply different methods to create three *TArray* objects in the program for working with various types

```
void OnStart()
  {
   TArray<double> double_array;   // vector has a default size of 10
   TArray<int> int_array(15);     // vector has a size of 15
   TArray<string> *string_array;  // pointer to TArray<string> vector
//--- create a dynamic object
   string_array=new TArray<string>(20);
//--- display an object name, data type and vector size in the Journal
   PrintFormat("%s (%s)",TOSTR(double_array),double_array.Type());
   PrintFormat("%s (%s)",TOSTR(int_array),int_array.Type());
   PrintFormat("%s (%s)",TOSTR(string_array),string_array.Type());
//--- remove a dynamic object before completing the program
   delete(string_array);
  }
```

Script execution results:

```
  double_array  (double:10)
  int_array  (int:15)
  string_array  (string:20)
```

Now, we have 3 vectors with different data types: double, int and string.

Class templates are well suited for developing containers – objects designed for encapsulating other objects of any type. Container objects are collections already containing objects of one certain type. Usually, working with stored data is instantly built into the container.

For example, you can create a class template that does not allow accessing an element outside the array, thus avoiding the "out of range" critical error.

```
//+------------------------------------------------------------------+
//| Class for a free access to an array element               |
//+------------------------------------------------------------------+
template<typename T>
class TSafeArray
  {
protected:
   T                 m_array[];
public:
   //--- default constructor
   void              TSafeArray(void){}
   //--- constructor for creating the array of a specified size
   void              TSafeArray(int size){ArrayResize(m_array,size);}
   //--- array size
   int               Size(void){return(ArraySize(m_array));}
   //--- change the array size
   int               Resize(int size,int reserve){return(ArrayResize(m_array,size,rese
   //--- release the array
   void              Erase(void){ZeroMemory(m_array);}
   //--- operator for accessing the array element by index
   T                 operator[](int index);
   //--- assignment operator for receiving all elements from the array at once
   void              operator=(const T  &array[]); // T type array
  };
//+------------------------------------------------------------------+
//| Receiving an element by index                             |
//+------------------------------------------------------------------+
template<typename T>
T TSafeArray::operator[](int index)
  {
   static T invalid_value;
//---
   int max=ArraySize(m_array)-1;
   if(index<0 || index>=ArraySize(m_array))
     {
      PrintFormat("%s index %d is not in range (0-%d)!",__FUNCTION__,index,max);
      return(invalid_value);
     }
//---
   return(m_array[index]);
  }
//+------------------------------------------------------------------+
//| Assigning for the array                                          |
//+------------------------------------------------------------------+
template<typename T>
void TSafeArray::operator=(const T  &array[])
  {
   int size=ArraySize(array);
   ArrayResize(m_array,size);
//--- T type should support the copying operator
   for(int i=0;i<size;i++)
      m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int copied,size=15;
   MqlRates rates[];
//--- copy the array of quotes
   if((copied=CopyRates(_Symbol,_Period,0,size,rates))!=size)
     {
      PrintFormat("CopyRates(%s,%s,0,%d) returned %d error code",
      _Symbol,EnumToString(_Period),size,GetLastError());
      return;
     }
//--- create a container and insert the MqlRates value array to it
   TSafeArray<MqlRates> safe_rates;
   safe_rates=rates;
   //--- index within the array
   int index=3;
   PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
   //--- index outside the array
   index=size;
   PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
  }
```

Please note that template declaration should also be used when describing methods outside the class declaration:

```
template<typename T>
T TSafeArray::operator[](int index)
  {
   ...
  }
template<typename T>
void TSafeArray::operator=(const T  &array[])
  {
   ...
   }
```

Class and function templates allow you to define multiple comma-separated formal parameters, for example, Map collection for storing "key – value" pairs:

```
template<typename Key, template Value>
class TMap
   {
    ...
   }
```

See also

Function templates, Overload

Abstract Classes and Pure Virtual Functions

Abstract classes are used for creating generic entities, that you expect to use for creating more specific derived classes. An abstract class can only be used as the base class for some other class, that is why it is impossible to create an object of the abstract class type.

A class which contains at least one pure virtual function in it is abstract. Therefore, classes derived from the abstract class must implement all its pure virtual functions, otherwise they will also be abstract classes.

A virtual function is declared as "pure" by using the pure-specifier syntax. Consider the example of the CAnimal class, which is only created to provide common functions – the objects of the CAnimal type are too general for practical use. Thus, CAnimal is a good example for an abstract class:

```
class CAnimal
  {
public:
                      CAnimal();     // Constructor
   virtual void       Sound() = 0;   // A pure virtual function
private:
   double             m_legs_count;  // The number of the animal's legs
  };
```

Here Sound() is a pure virtual function, because it is declared with the specifier of the pure virtual function PURE (**=0**).

Pure virtual functions are only the virtual functions for which the PURE specifier is set: (=NULL) or (=0). Example of abstract class declaration and use:

```
class CAnimal
  {
public:
   virtual void       Sound()=NULL;   // PURE method, should be overridden in the der
  };
//--- Derived from an abstract class
class CCat : public CAnimal
 {
public:
  virtual void        Sound() { Print("Myau"); } // PURE is overridden, CCat is not ab
 };

//--- Examples of wrong use
new CAnimal;         // Error of 'CAnimal' - the compiler returns the "cannot instant
CAnimal some_animal; // Error of 'CAnimal' - the compiler returns the "cannot instant

//--- Examples of proper use
new CCat;  // No error - the CCat class is not abstract
CCat cat;  // No error - the CCat class is not abstract
```

Restrictions on abstract classes

If the constructor for an abstract class calls a pure virtual function (either directly or indirectly), the result is undefined.

```
//+------------------------------------------------------------------+
//| An abstract base class                                           |
//+------------------------------------------------------------------+
class CAnimal
  {
public:
   //--- A pure virtual function
   virtual void      Sound(void)=NULL;
   //--- Function
   void              CallSound(void) { Sound(); }
   //--- Constructor
   CAnimal()
    {
     //--- An explicit call of the virtual method
     Sound();
     //--- An implicit call (using a third function)
     CallSound();
     //--- A constructor and/or destructor always calls its own functions,
     //--- even if they are virtual and overridden by a called function in a derived 
     //--- If the called function is pure virtual,
     //--- its call will cause a critical runtime error: "pure virtual function call"
    }
  };
```

However, constructors and destructors for abstract classes can call other member functions.

Namespaces

A namespace is a specially declared area, within which various IDs are defined: variables, functions, classes, etc. It is set using the namespace keyword:

```
namespace name of_space {
  // list of function, class and variable definitions
}
```

Applying 'namespace' allows splitting the global namespace into subspaces. All IDs within the namespace are available to each other without a specification. The :: operator (context resolution operation) is used to access namespace members from the outside.

```
namespace ProjectData
{
class DataManager
  {
public:
   void              LoadData() {}
  };
void Func(DataManager& manager) {}
}
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- working with ProjectData namespace
   ProjectData::DataManager mgr;
   mgr.LoadData();
   ProjectData::Func(mgr);
  }
```

Namespaces are used to arrange a code in the form of logical groups and to avoid name conflicts that may occur when several libraries are used in a program. In such cases, each library can be declared in its namespace to explicitly access the necessary functions and classes of each library.

A namespace can be declared in several blocks in one or several files. The compiler combines all parts together during a preprocessing and the resulting namespace contains all the members declared in all parts. Suppose that we have A class implemented in the Sample.mqh include file:

```
//+------------------------------------------------------------------+
//|                                                       Sample.mqh |
//+------------------------------------------------------------------+
class A
  {
public:
                     A() {Print(__FUNCTION__);}
  };
```

We want to use this class in our project, but we already have A class. To be able to use both classes and avoid ID conflict, simply wrap the included file in a namespace:

```
//--- declare the first A class
class A
  {
public:
                     A() {Print(__FUNCTION__);}
  };
//--- wrap the A class from the Sample.mqh file in the Library namespace to avoid a co
namespace Library
{
#include "Sample.mqh"
}
//--- add yet another class to the Library namespace
namespace Library
{
class B
  {
public:
                     B() {Print(__FUNCTION__);}
  };
}
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- use the A class from the global namespace
   A a1;
//--- use the A and B classes from the Library namespace
   Library::A a2;
   Library::B b;
  }
//+------------------------------------------------------------------+

/*
Result:
   A::A
   Library::A::A
   Library::B::B
*/
```

Namespaces can be nested. A nested namespace has unlimited access to members of its parent space, but members of the parent space do not have unlimited access to the nested namespace.

```
namespace General
{
int Func();
namespace Details
{
 int Counter;
 int Refresh()  {return Func(); }
}

int GetBars()   {return(iBars(Symbol(), Period()));};
int Size(int i) {return Details::Counter;}
}
```

Global namespace

If the ID is not explicitly declared in the namespace, it is considered to be an implicit part of the global namespace. To set the global ID explicitly, use the scope resolution operator without a name. This will allow you to distinguish this ID from any other element with the same name located in a different namespace. For example, when importing a function:

```
#import "lib.dll"
int Func();
#import
//+------------------------------------------------------------------+
//|  Some function                                                  |
//+------------------------------------------------------------------+
int Func()
  {
   return(0);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//+--- call the imported function
   Print(lib::Func());
//+--- call our function
   Print(::Func());
  }
```

In this case, all the functions imported from the DLL function were included in the namespace of the same name. This allowed the compiler to clearly determine the function to be called.

See also

Global Variables, Local Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Constants, Enumerations and Structures

To simplify the program writing and to make program texts more convenient for perception, the MQL5 language provides predefined standard constants and enumerations. Besides that, service structures are used for storing information.

Standard constants are similar to macros and are of int type.

The constants are grouped by their purposes:

- Chart constants are used when working with price charts: opening, navigation, setting parameters;
- Objects constants are intended for processing graphical objects that can be created and displayed in charts;
- Indicators constants are used for working with standard and custom indicators;
- Environment state constants describe properties of a MQL5-program, show information about a client terminal, financial instrument and current account;
- Trade constants allow to specify a variety of information in the course of trading;
- Named constants are constants of the MQL5 language;
- Data structures describe data storage formats used;
- Codes of errors and warnings describe compiler messages and trading server answers to trade requests;
- In/out constants are designed for working with file functions and displaying messages on the screen by the MessageBox() function.

Chart Constants

Constants describing various properties of charts are divided into the following groups:

- Types of events – events that occur when working with charts;
- Chart timeframes – standard built-in periods;
- Properties of chart – identifiers that are used as parameters of chart functions;
- Positioning constants - value of a parameter of the ChartNavigate() function;
- Displaying charts - setting the chart appearance.

Types of Chart Events

There are 11 types of events that can be processed using the predefined function OnChartEvent(). For custom events 65535 identifiers are provided in the range of CHARTEVENT_CUSTOM to CHARTEVENT_CUSTOM_LAST inclusive. To generate a custom event, the EventChartCustom() function should be used.

ENUM_CHART_EVENT

ID	Description
CHARTEVENT_KEYDOWN	Keystrokes
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)
CHARTEVENT_MOUSE_WHEEL	Pressing or scrolling the mouse wheel (if CHART_EVENT_MOUSE_WHEEL=True for the chart)
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)
CHARTEVENT_CLICK	Clicking on a chart
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit
CHARTEVENT_CHART_CHANGE	Change of the chart size or modification of chart properties through the Properties dialog
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through this parameters are listed in the below table.

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of times the keystroke is repeated as a result of the user holding down the key)	The string value of a bit mask describing the status of keyboard buttons
Mouse events (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
Mouse wheel event (if CHART_EVENT_MOUSE_WHEEL=true for the chart)	CHARTEVENT_MOUSE_WHEEL	Flags of states of keys and mouse buttons, the X and Y coordinates of the mouse pointer. See description in the example below	The Delta value of the mouse wheel scroll	—
event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object
Event of a mouse click on	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
the chart				
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit graphical object, in which text editing has completed
Event of change of the chart size or modification of chart properties through the Properties dialog	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

Example:

```
#define KEY_NUMPAD_5       12
#define KEY_LEFT           37
#define KEY_UP             38
#define KEY_RIGHT          39
#define KEY_DOWN           40
#define KEY_NUMLOCK_DOWN   98
#define KEY_NUMLOCK_LEFT  100
#define KEY_NUMLOCK_5     101
#define KEY_NUMLOCK_RIGHT 102
#define KEY_NUMLOCK_UP    104
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
```

```
  {
//---
   Print("The expert with name ",MQL5InfoString(MQL5_PROGRAM_NAME)," is running");
//--- enable object create events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- enable object delete events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//--- forced updating of chart properties ensures readiness for event processing
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // Event identifier
                  const long& lparam,   // Event parameter of long type
                  const double& dparam, // Event parameter of double type
                  const string& sparam  // Event parameter of string type
                  )
  {
//--- the left mouse button has been pressed on the chart
   if(id==CHARTEVENT_CLICK)
     {
      Print("The coordinates of the mouse click on the chart are: x = ",lparam,"  y =
     }
//--- the mouse has been clicked on the graphic object
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("The mouse has been clicked on the object with name '"+sparam+"'");
     }
//--- the key has been pressed
   if(id==CHARTEVENT_KEYDOWN)
     {
      switch(lparam)
        {
         case KEY_NUMLOCK_LEFT:  Print("The KEY_NUMLOCK_LEFT has been pressed");   bre
         case KEY_LEFT:          Print("The KEY_LEFT has been pressed");           bre
         case KEY_NUMLOCK_UP:    Print("The KEY_NUMLOCK_UP has been pressed");     bre
         case KEY_UP:            Print("The KEY_UP has been pressed");             bre
         case KEY_NUMLOCK_RIGHT: Print("The KEY_NUMLOCK_RIGHT has been pressed");  bre
         case KEY_RIGHT:         Print("The KEY_RIGHT has been pressed");          bre
         case KEY_NUMLOCK_DOWN:  Print("The KEY_NUMLOCK_DOWN has been pressed");   bre
         case KEY_DOWN:          Print("The KEY_DOWN has been pressed");           bre
         case KEY_NUMPAD_5:      Print("The KEY_NUMPAD_5 has been pressed");       bre
         case KEY_NUMLOCK_5:     Print("The KEY_NUMLOCK_5 has been pressed");      bre
         default:                Print("Some not listed key has been pressed");
        }
      ChartRedraw();
     }
//--- the object has been deleted
   if(id==CHARTEVENT_OBJECT_DELETE)
     {
      Print("The object with name ",sparam," has been deleted");
     }
//--- the object has been created
   if(id==CHARTEVENT_OBJECT_CREATE)
     {
      Print("The object with name ",sparam," has been created");
     }
//--- the object has been moved or its anchor point coordinates has been changed
   if(id==CHARTEVENT_OBJECT_DRAG)
     {
      Print("The anchor point coordinates of the object with name ",sparam," has been
     }
//--- the text in the Edit of object has been changed
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
     {
      Print("The text in the Edit field of the object with name ",sparam," has been ch
     }
  }
```

For CHARTEVENT_MOUSE_MOVE event the **sparam** string parameter contains information about state of the keyboard and mouse buttons:

Bit	Description
1	State of the left mouse button
2	State of the right mouse button
3	State of the SHIFT button
4	State of the CTRL button
5	State of the middle mouse button
6	State of the first extra mouse button
7	State of the second extra mouse button

Example:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- enable CHART_EVENT_MOUSE_MOVE messages
   ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
//--- disable the context menu of the chart (on the right)
```

```
   ChartSetInteger(0,CHART_CONTEXT_MENU,0);
//--- disable the crosshair (by the middle button)
   ChartSetInteger(0,CHART_CROSSHAIR_TOOL,0);
//--- forced updating of chart properties ensures readiness for event processing
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| MouseState                                                       |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // mouse left
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // mouse right
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // mouse middle
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // mouse first X key
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // mouse second X key
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // shift key
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // control key
   return(res);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
  {
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
  }
```

For the CHARTEVENT_MOUSE_WHEEL event, parameters **lparam** and **dparam** contain information about the states of the **Ctrl** and **Shift** keys, of mouse buttons, cursor coordinates and the mouse wheel scroll value. For a better understanding, run this Expert Advisor on a chart and scroll the mouse wheel, while pressing different buttons and holding down the keys described in the code.

Example of CHARTEVENT_MOUSE_WHEEL event processing:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
init OnInit()
  {
//--- Enabling mouse wheel scrolling messages
   ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
//--- Forced updating of chart properties ensures readiness for event processing
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &
  {
   if(id==CHARTEVENT_MOUSE_WHEEL)
     {
      //--- Consider the state of mouse buttons and wheel for this event
      int flg_keys = (int)(lparam>>32);          // The flag of states of the Ctrl an
      int x_cursor = (int)(short)lparam;         // the X coordinate where the mousse
      int y_cursor = (int)(short)(lparam>>16);   // the Y coordinate where the mousse
      int delta    = (int)dparam;                // the total value of mouse scroll,
      //--- Processing the flag
      string str_keys="";
      if((flg_keys&0x0001)!=0) str_keys+="LMOUSE ";
      if((flg_keys&0x0002)!=0) str_keys+="RMOUSE ";
      if((flg_keys&0x0004)!=0) str_keys+="SHIFT ";
      if((flg_keys&0x0008)!=0) str_keys+="CTRL ";
      if((flg_keys&0x0010)!=0) str_keys+="MMOUSE ";
      if((flg_keys&0x0020)!=0) str_keys+="X1MOUSE ";
      if((flg_keys&0x0040)!=0) str_keys+="X2MOUSE ";

      if(str_keys!="")
         str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1) + "'";
      PrintFormat("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x_
     }
  }
//+------------------------------------------------------------------+ /*
   Example of the output
   CHARTEVENT_MOUSE_WHEEL: Ctrl pressed: X=193, Y=445, delta=-120
   CHARTEVENT_MOUSE_WHEEL: Shift pressed: X=186, Y=446, delta=120
   CHARTEVENT_MOUSE_WHEEL:  X=178, Y=447, delta=-120
   CHARTEVENT_MOUSE_WHEEL:  X=231, Y=449, delta=120
   CHARTEVENT_MOUSE_WHEEL: MiddleButton pressed: X=231, Y=449, delta=120
   CHARTEVENT_MOUSE_WHEEL: LeftButton pressed: X=279, Y=320, delta=-120
   CHARTEVENT_MOUSE_WHEEL: RightButton pressed: X=253, Y=330, delta=120  */
```

See also

Event Handling Functions, Working with events

Chart Timeframes

All predefined timeframes of charts have unique identifiers. The PERIOD_CURRENT identifier means the current period of a chart, at which a mql5-program is running.

ENUM_TIMEFRAMES

ID	Description
PERIOD_CURRENT	Current timeframe
PERIOD_M1	1 minute
PERIOD_M2	2 minutes
PERIOD_M3	3 minutes
PERIOD_M4	4 minutes
PERIOD_M5	5 minutes
PERIOD_M6	6 minutes
PERIOD_M10	10 minutes
PERIOD_M12	12 minutes
PERIOD_M15	15 minutes
PERIOD_M20	20 minutes
PERIOD_M30	30 minutes
PERIOD_H1	1 hour
PERIOD_H2	2 hours
PERIOD_H3	3 hours
PERIOD_H4	4 hours
PERIOD_H6	6 hours
PERIOD_H8	8 hours
PERIOD_H12	12 hours
PERIOD_D1	1 day
PERIOD_W1	1 week
PERIOD_MN1	1 month

Example:

```
   string chart_name="test_Object_Chart";
   Print("Let's try to create a Chart object with the name ",chart_name);
//--- If such an object does not exist - create it
   if(ObjectFind(0,chart_name)<0)ObjectCreate(0,chart_name,OBJ_CHART,0,0,0,0,0);
```

```
//--- Define symbol
   ObjectSetString(0,chart_name,OBJPROP_SYMBOL,"EURUSD");
//--- Set X coordinate of the anchor point
   ObjectSetInteger(0,chart_name,OBJPROP_XDISTANCE,100);
//--- Set Y coordinate of the anchor point
   ObjectSetInteger(0,chart_name,OBJPROP_YDISTANCE,100);
//--- Set the width of chart
   ObjectSetInteger(0,chart_name,OBJPROP_XSIZE,400);
//--- Set the height
   ObjectSetInteger(0,chart_name,OBJPROP_YSIZE,300);
//--- Set the timeframe
   ObjectSetInteger(0,chart_name,OBJPROP_PERIOD,PERIOD_D1);
//--- Set scale (from 0 to 5)
   ObjectSetDouble(0,chart_name,OBJPROP_SCALE,4);
//--- Disable selection by a mouse
   ObjectSetInteger(0,chart_name,OBJPROP_SELECTABLE,false);
```

Timeseries identifiers

The identifiers of timeseries are used in the iHighest() and iLowest() functions. They can be equal to a value the enumeration

ENUM_SERIESMODE

Identifier	Description
MODE_OPEN	Opening price
MODE_LOW	Low price
MODE_HIGH	High price
MODE_CLOSE	Close price
MODE_VOLUME	Tick volume
MODE_REAL_VOLUME	Real volume
MODE_SPREAD	Spread

See also

PeriodSeconds, Period, Date and Time, Visibility of objects

Chart Properties

Identifiers of ENUM_CHART_PROPERTY enumerations are used as parameters of functions for working with charts. The abbreviation of r/o in the "Property Type" column means that this property is read-only and cannot be changed. The w/o abbreviation in the "Property Type" column means that this property is write-only and it cannot be received. When accessing certain properties, it's necessary to specify an additional parameter-modifier (modifier), which serves to indicate the number of chart subwindows. 0 means the main window.

The functions defining the chart properties are actually used for sending change commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. The changes are implemented to the chart when handling the queue of the chart events.

Thus, do not expect an immediate visual update of the chart after calling these functions. Generally, the chart is updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use ChartRedraw() function to forcefully update the chart.

For functions ChartSetInteger() and ChartGetInteger()

ENUM_CHART_PROPERTY_INTEGER

ID	Description	Property Type
CHART_SHOW	Price chart drawing. If false, drawing any price chart attributes is disabled and all chart border indents are eliminated, including time and price scales, quick navigation bar, Calendar event labels, trade labels, indicator and bar tooltips, indicator subwindows, volume histograms, etc. Disabling the drawing is a perfect solution for creating a custom program interface using the graphical resources. The graphical objects are always drawn regardless of the CHART_SHOW property value.	bool
CHART_IS_OBJECT	Identifying "Chart" (OBJ_CHART) object – returns true for a graphical object. Returns false for a real chart	bool r/o
CHART_BRING_TO_TOP	Show chart on top of other charts	bool
CHART_CONTEXT_MENU	Enabling/disabling access to the context menu using the right click. When CHART_CONTEXT_MENU=false, only the chart context menu is disabled. The context menu of objects on the chart remains available.	bool (default is true)

ID	Description	Property Type
CHART_CROSSHAIR_TOOL	Enabling/disabling access to the Crosshair tool using the middle click.	bool (default is true)
CHART_MOUSE_SCROLL	Scrolling the chart horizontally using the left mouse button. Vertical scrolling is also available if the value of any following properties is set to true: CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR When CHART_MOUSE_SCROLL=false, chart scrolling with the mouse wheel is unavailable	bool
CHART_EVENT_MOUSE_WHEEL	Sending messages about mouse wheel events (CHARTEVENT_MOUSE_WHEEL) to all mql5 programs on a chart	bool (default is true)
CHART_EVENT_MOUSE_MOVE	Send notifications of mouse move and mouse click events (CHARTEVENT_MOUSE_MOVE) to all mql5 programs on a chart	bool
CHART_EVENT_OBJECT_CREATE	Send a notification of an event of new object creation (CHARTEVENT_OBJECT_CREATE) to all mql5-programs on a chart	bool
CHART_EVENT_OBJECT_DELETE	Send a notification of an event of object deletion (CHARTEVENT_OBJECT_DELETE) to all mql5-programs on a chart	bool
CHART_MODE	Chart type (candlesticks, bars or line)	enum ENUM_CHART_MODE
CHART_FOREGROUND	Price chart in the foreground	bool
CHART_SHIFT	Mode of price chart indent from the right border	bool
CHART_AUTOSCROLL	Mode of automatic moving to the right border of the chart	bool
CHART_KEYBOARD_CONTROL	Allow managing the chart using a keyboard ("Home", "End", "PageUp", "+", "-", "Up arrow", etc.). Setting CHART_KEYBOARD_CONTROL to false disables chart scrolling and scaling while leaving intact the ability to receive the keys pressing events in OnChartEvent().	bool
CHART_QUICK_NAVIGATION	Allow the chart to intercept Space and Enter key strokes to activate the quick navigation bar. The quick navigation bar	bool

ID	Description	Property Type
	automatically appears at the bottom of the chart after double-clicking the mouse or pressing Space/Enter. It allows you to quickly change a symbol, timeframe and first visible bar date.	
CHART_SCALE	Scale	int from 0 to 5
CHART_SCALEFIX	Fixed scale mode	bool
CHART_SCALEFIX_11	Scale 1:1 mode	bool
CHART_SCALE_PT_PER_BAR	Scale to be specified in points per bar	bool
CHART_SHOW_TICKER	Display a symbol ticker in the upper left corner. Setting CHART_SHOW_TICKER to 'false' also sets CHART_SHOW_OHLC to 'false' and disables OHLC	bool
CHART_SHOW_OHLC	Display OHLC values in the upper left corner. Setting CHART_SHOW_OHLC to 'true' also sets CHART_SHOW_TICKER to 'true' and enables the ticker	bool
CHART_SHOW_BID_LINE	Display Bid values as a horizontal line in a chart	bool
CHART_SHOW_ASK_LINE	Display Ask values as a horizontal line in a chart	bool
CHART_SHOW_LAST_LINE	Display Last values as a horizontal line in a chart	bool
CHART_SHOW_PERIOD_SEP	Display vertical separators between adjacent periods	bool
CHART_SHOW_GRID	Display grid in the chart	bool
CHART_SHOW_VOLUMES	Display volume in the chart	enum ENUM_CHART_VOLUME_MODE
CHART_SHOW_OBJECT_DESCR	Display textual descriptions of objects (not available for all objects)	bool
CHART_SHOW_TRADE_HISTORY	Display trades from the trading history as entry/exit arrows on a chart. See the "Show trading history" option descriptions in the platform settings.	bool
CHART_VISIBLE_BARS	The number of bars on the chart that can be displayed	int r/o

ID	Description	Property Type
CHART_WINDOWS_TOTAL	The total number of chart windows, including indicator subwindows	int r/o
CHART_WINDOW_IS_VISIBLE	Visibility of subwindows	bool r/o modifier - subwindow number
CHART_WINDOW_HANDLE	Chart window handle (HWND)	int r/o
CHART_WINDOW_YDISTANCE	The distance between the upper frame of the indicator subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a mouse event, the cursor coordinates are passed in terms of the coordinates of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow. The value is required for converting the absolute coordinates of the main chart to the local coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.	int r/o modifier - subwindow number
CHART_FIRST_VISIBLE_BAR	Number of the first visible bar in the chart. Indexing of bars is the same as for timeseries.	int r/o
CHART_WIDTH_IN_BARS	Chart width in bars	int r/o
CHART_WIDTH_IN_PIXELS	Chart width in pixels	int r/o
CHART_HEIGHT_IN_PIXELS	Chart height in pixels	int modifier - subwindow number
CHART_COLOR_BACKGROUND	Chart background color	color
CHART_COLOR_FOREGROUND	Color of axes, scales and OHLC line	color
CHART_COLOR_GRID	Grid color	color
CHART_COLOR_VOLUME	Color of volumes and position opening levels	color
CHART_COLOR_CHART_UP	Color for the up bar, shadows and body borders of bull candlesticks	color

ID	Description	Property Type
CHART_COLOR_CHART_DOWN	Color for the down bar, shadows and body borders of bear candlesticks	color
CHART_COLOR_CHART_LINE	Line chart color and color of "Doji" Japanese candlesticks	color
CHART_COLOR_CANDLE_BULL	Body color of a bull candlestick	color
CHART_COLOR_CANDLE_BEAR	Body color of a bear candlestick	color
CHART_COLOR_BID	Bid price level color	color
CHART_COLOR_ASK	Ask price level color	color
CHART_COLOR_LAST	Line color of the last executed deal price (Last)	color
CHART_COLOR_STOP_LEVEL	Color of stop order levels (Stop Loss and Take Profit)	color
CHART_SHOW_TRADE_LEVELS	Displaying trade levels in the chart (levels of open positions, Stop Loss, Take Profit and pending orders)	bool
CHART_DRAG_TRADE_LEVELS	Permission to drag trading levels on a chart with a mouse. The drag mode is enabled by default (true value)	bool
CHART_SHOW_DATE_SCALE	Showing the time scale on a chart	bool
CHART_SHOW_PRICE_SCALE	Showing the price scale on a chart	bool
CHART_SHOW_ONE_CLICK	Showing the "One click trading" panel on a chart	bool
CHART_IS_MAXIMIZED	Chart window is maximized	bool r/o
CHART_IS_MINIMIZED	Chart window is minimized	bool r/o
CHART_IS_DOCKED	The chart window is docked. If set to false, the chart can be dragged outside the terminal area	bool
CHART_FLOAT_LEFT	The left coordinate of the undocked chart window relative to the virtual screen	int
CHART_FLOAT_TOP	The top coordinate of the undocked chart window relative to the virtual screen	int
CHART_FLOAT_RIGHT	The right coordinate of the undocked chart window relative to the virtual screen	int

ID	Description	Property Type
CHART_FLOAT_BOTTOM	The bottom coordinate of the undocked chart window relative to the virtual screen	int

For functions ChartSetDouble() and ChartGetDouble()

ENUM_CHART_PROPERTY_DOUBLE

ID	Description	Property Type
CHART_SHIFT_SIZE	The size of the zero bar indent from the right border in percents	double (from 10 to 50 percents)
CHART_FIXED_POSITION	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the horizontal time axis. It is displayed only if the automatic chart scrolling to the right on tick incoming is disabled (see CHART_AUTOSCROLL property). The bar on a fixed position remains in the same place when zooming in and out.	double
CHART_FIXED_MAX	Fixed chart maximum	double
CHART_FIXED_MIN	Fixed chart minimum	double
CHART_POINTS_PER_BAR	Scale in points per bar	double
CHART_PRICE_MIN	Chart minimum	double r/o modifier - subwindow number
CHART_PRICE_MAX	Chart maximum	double r/o modifier - subwindow number

For functions ChartSetString() and ChartGetString()

ENUM_CHART_PROPERTY_STRING

ID	Description	Property Type
CHART_COMMENT	Text of a comment in a chart	string
CHART_EXPERT_NAME	The name of the Expert Advisor running on the chart with the specified chart_id	string r/o
CHART_SCRIPT_NAME	The name of the script running on the chart with the specified chart_id	string r/o

Example:

```
   int chartMode=ChartGetInteger(0,CHART_MODE);
   switch(chartMode)
     {
      case(CHART_BARS):    Print("CHART_BARS");   break;
      case(CHART_CANDLES): Print("CHART_CANDLES");break;
      default:Print("CHART_LINE");
     }
   bool shifted=ChartGetInteger(0,CHART_SHIFT);
   if(shifted) Print("CHART_SHIFT = true");
   else Print("CHART_SHIFT = false");
   bool autoscroll=ChartGetInteger(0,CHART_AUTOSCROLL);
   if(autoscroll) Print("CHART_AUTOSCROLL = true");
   else Print("CHART_AUTOSCROLL = false");
   int chartHandle=ChartGetInteger(0,CHART_WINDOW_HANDLE);
   Print("CHART_WINDOW_HANDLE = ",chartHandle);
   int windows=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   Print("CHART_WINDOWS_TOTAL = ",windows);
   if(windows>1)
     {
      for(int i=0;i<windows;i++)
        {
         int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,i);
         double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,i);
         double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,i);
         Print(i+": CHART_HEIGHT_IN_PIXELS = ",height," pixels");
         Print(i+": CHART_PRICE_MIN = ",priceMin);
         Print(i+": CHART_PRICE_MAX = ",priceMax);
        }
        }
```

See also

[Examples of Working with the Chart]

Positioning Constants

Three identifiers from the ENUM_CHART_POSITION list are the possible values of the *position* parameter for the ChartNavigate() function.

ENUM_CHART_POSITION

ID	Description
CHART_BEGIN	Chart beginning (the oldest prices)
CHART_CURRENT_POS	Current position
CHART_END	Chart end (the latest prices)

Example:

```
   long handle=ChartOpen("EURUSD",PERIOD_H12);
   if(handle!=0)
     {
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      ChartSetInteger(handle,CHART_SHIFT,true);
      ChartSetInteger(handle,CHART_MODE,CHART_LINE);
      ResetLastError();
      bool res=ChartNavigate(handle,CHART_END,150);
      if(!res) Print("Navigate failed. Error = ",GetLastError());
      ChartRedraw();
     }
```

Chart Representation

Price charts can be displayed in three ways:

- as bars;
- as candlesticks;
- as a line.

The specific way of displaying the price chart is set by the function ChartSetInteger(chart_handle,CHART_MODE, chart_mode), where chart_mode is one of the values of the ENUM_CHART_MODE enumeration.

ENUM_CHART_MODE

ID	Description
CHART_BARS	Display as a sequence of bars
CHART_CANDLES	Display as Japanese candlesticks
CHART_LINE	Display as a line drawn by Close prices

To specify the mode of displaying volumes in the price chart the function ChartSetInteger(chart_handle, CHART_SHOW_VOLUMES, volume_mode) is used, where volume_mode is one of values of the ENUM_CHART_VOLUME_MODE enumeration.

ENUM_CHART_VOLUME_MODE

ID	Description
CHART_VOLUME_HIDE	Volumes are not shown
CHART_VOLUME_TICK	Tick volumes
CHART_VOLUME_REAL	Trade volumes

Example:

```
//--- Get the handle of the current chart
   long handle=ChartID();
   if(handle>0) // If it succeeded, additionally customize
     {
      //--- Disable autoscroll
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- Set the indent of the right border of the chart
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- Display as candlesticks
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- Scroll by 100 bars from the beginning of history
      ChartNavigate(handle,CHART_CURRENT_POS,100);
      //--- Set the tick volume display mode
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
```

```
        }
```

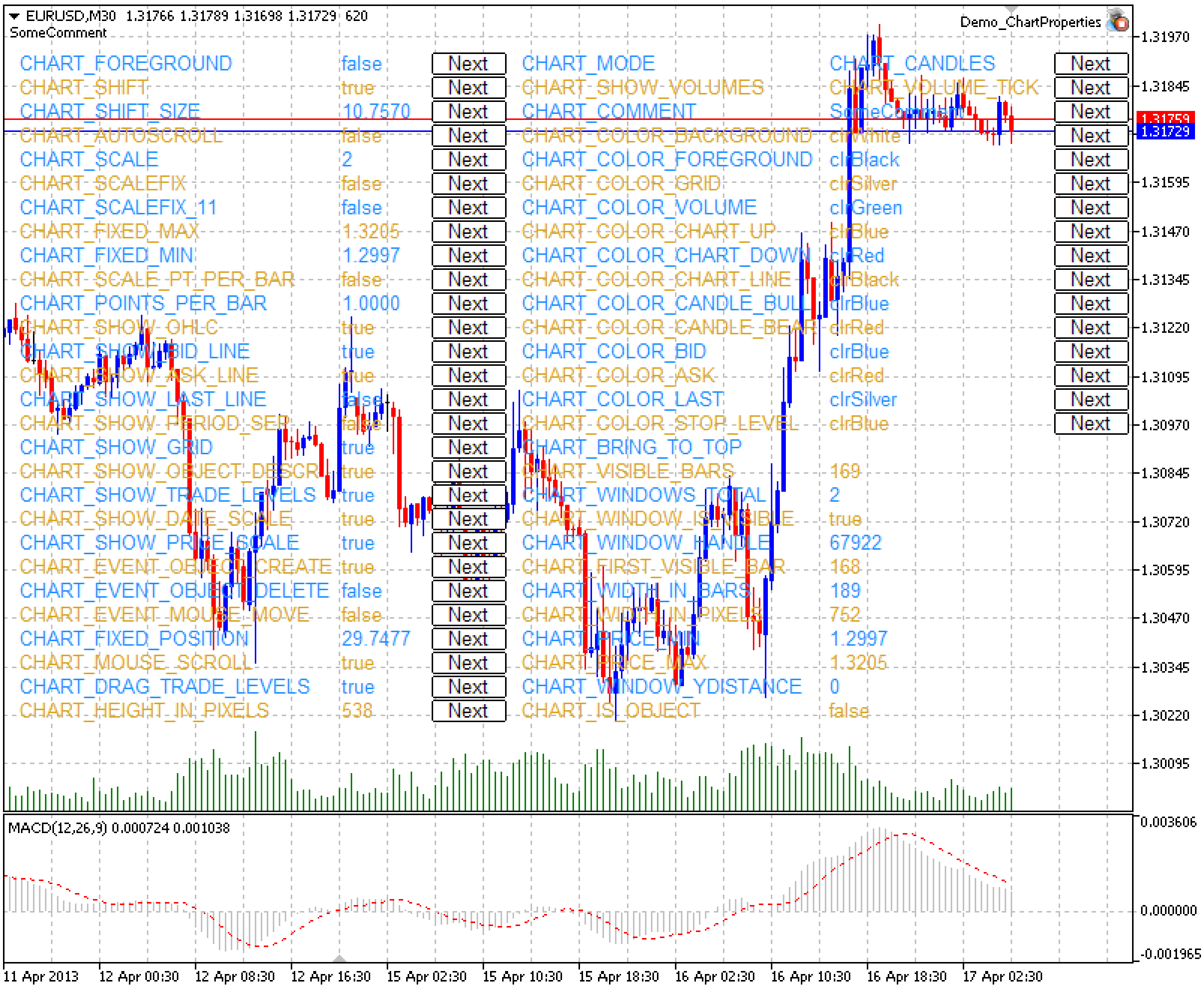
See also

[ChartOpen](), [ChartID]()

Examples of Working with the Chart

This section contains examples of working with chart properties. One or two complete functions are displayed for each property. These functions allow setting/receiving the value of the property. These functions can be used "as is" in custom mql5 applications.

The screenshot below demonstrates the graphic panel illustrating how changing of the chart property changes its appearance. Clicking Next button allows setting the new value of the appropriate property and view the changes in the chart window.


Demo_ChartProperties
Next
1.31970
1.31845
1.31759
1.31729
1.31595
1.31470
1.31345
1.31220
1.31095
1.30970
1.30845
1.30720
1.30595
1.30470
1.30345
1.30220
1.30095
0.003606
0.000000
-0.001965
11 Apr 2013
12 Apr 00:30
12 Apr 08:30
12 Apr 16:30
15 Apr 02:30
15 Apr 10:30
15 Apr 18:30
16 Apr 02:30
16 Apr 10:30
16 Apr 18:30
17 Apr 02:30

EURUSD,M30 1.31766 1.31789 1.31698 1.31729 620
SomeComment

MACD(12,26,9) 0.000724 0.001038

CHART_FOREGROUND	false	CHART_MODE	CHART_CANDLES
CHART_SHIFT	true	CHART_SHOW_VOLUMES	CHART_VOLUME_TICK
CHART_SHIFT_SIZE	10.7570	CHART_COMMENT	SomeComment
CHART_AUTOSCROLL	false	CHART_COLOR_BACKGROUND	clrWhite
CHART_SCALE	2	CHART_COLOR_FOREGROUND	clrBlack
CHART_SCALEFIX	false	CHART_COLOR_GRID	clrSilver
CHART_SCALEFIX_11	false	CHART_COLOR_VOLUME	clrGreen
CHART_FIXED_MAX	1.3205	CHART_COLOR_CHART_UP	clrBlue
CHART_FIXED_MIN	1.2997	CHART_COLOR_CHART_DOWN	clrRed
CHART_SCALE_PT_PER_BAR	false	CHART_COLOR_CHART_LINE	clrBlack
CHART_POINTS_PER_BAR	1.0000	CHART_COLOR_CANDLE_BULL	clrBlue
CHART_SHOW_OHLC	true	CHART_COLOR_CANDLE_BEAR	clrRed
CHART_SHOW_BID_LINE	true	CHART_COLOR_BID	clrBlue
CHART_SHOW_ASK_LINE	true	CHART_COLOR_ASK	clrRed
CHART_SHOW_LAST_LINE	false	CHART_COLOR_LAST	clrSilver
CHART_SHOW_PERIOD_SEP	false	CHART_COLOR_STOP_LEVEL	clrBlue
CHART_SHOW_GRID	true	CHART_BRING_TO_TOP	
CHART_SHOW_OBJECT_DESCR	true	CHART_VISIBLE_BARS	169
CHART_SHOW_TRADE_LEVELS	true	CHART_WINDOWS_TOTAL	2
CHART_SHOW_DATE_SCALE	true	CHART_WINDOW_IS_VISIBLE	true
CHART_SHOW_PRICE_SCALE	true	CHART_WINDOW_HANDLE	67922
CHART_EVENT_OBJECT_CREATE	true	CHART_FIRST_VISIBLE_BAR	168
CHART_EVENT_OBJECT_DELETE	false	CHART_WIDTH_IN_BARS	189
CHART_EVENT_MOUSE_MOVE	false	CHART_WIDTH_IN_PIXELS	752
CHART_FIXED_POSITION	29.7477	CHART_PRICE_MIN	1.2997
CHART_MOUSE_SCROLL	true	CHART_PRICE_MAX	1.3205
CHART_DRAG_TRADE_LEVELS	true	CHART_WINDOW_YDISTANCE	0
CHART_HEIGHT_IN_PIXELS	538	CHART_IS_OBJECT	false

The panel's source code is located below.

Chart Properties and Sample Functions for Working with Them

- **CHART_IS_OBJECT** defines if an object is a real chart or a graphic object.

```
//+------------------------------------------------------------------+
//| Checks if an object is a chart. If it is a graphic object,       |
//| the result is true. If it is a real chart, the result variable   |
//| has the value of false.                                          |
//+------------------------------------------------------------------+
bool ChartIsObject(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- get the chart property
   if(!ChartGetInteger(chart_ID,CHART_IS_OBJECT,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      //--- return false
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
```

- **CHART_BRING_TO_TOP** shows the chart on top of all others.

```
//+------------------------------------------------------------------+
//| Sends command to the terminal to display the chart above all others  |
//+------------------------------------------------------------------+
bool ChartBringToTop(const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- show the chart on top of all others
   if(!ChartSetInteger(chart_ID,CHART_BRING_TO_TOP,0,true))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_MOUSE_SCROLL** is a property for scrolling the chart using left mouse button.

```
//+------------------------------------------------------------------------------+
//| Checks if scrolling of chart using left mouse button is enabled              |
//+------------------------------------------------------------------------------+
bool ChartMouseScrollGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables scrolling of chart using left mouse button      |
//+------------------------------------------------------------------+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_EVENT_MOUSE_MOVE** is a property of sending messages concerning move events and mouse clicks to mql5 applications (CHARTEVENT_MOUSE_MOVE).

```
//+------------------------------------------------------------------+
//| Checks if messages concerning move events and mouse clicks       |
//| are sent to all MQL5 applications on the chart                   |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the mode of sending messages concerning move events and     |
//| mouse clicks to MQL5 applications on the chart                   |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_EVENT_OBJECT_CREATE** is a property of sending messages concerning the event of a graphic object creation to mql5 applications (CHARTEVENT_OBJECT_CREATE).

```
//+------------------------------------------------------------------+
//| Checks if messages concerning the event of a graphic             |
//| object creation are sent to all MQL5 applications on the chart   |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the mode of sending messages concerning the event of a  |
//| graphic object creation to all mql5 applications on the chart            |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_EVENT_OBJECT_DELETE** is a property of sending messages concerning the event of a graphic object deletion to mql5 applications (CHARTEVENT_OBJECT_DELETE).

```
//+------------------------------------------------------------------+
//| Checks if messages concerning the event of a graphic object      |
//| deletion are sent to all mql5 applications on the chart          |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
```

```
      {
       //--- display the error message in Experts journal
       Print(__FUNCTION__+", Error Code = ",GetLastError());
       return(false);
      }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the mode of sending messages concerning the event of a  |
//| graphic object deletion to all mql5 applications on the chart            |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_MODE** - type of the chart (candlesticks, bars or line).

```
//+------------------------------------------------------------------+
//| Gets chart display type (candlesticks, bars or line)             |
//+------------------------------------------------------------------+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=WRONG_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((ENUM_CHART_MODE)result);
  }
//+------------------------------------------------------------------+
//| Sets chart display type (candlesticks, bars or line)             |
//+------------------------------------------------------------------+
bool ChartModeSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_MODE,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FOREGROUND** is a property of displaying a price chart in the foreground.

```
//+------------------------------------------------------------------+
//| Checks if a price chart is displayed in the foreground           |
//+------------------------------------------------------------------+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of a price chart on the foreground   |
//+------------------------------------------------------------------+
bool ChartForegroundSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
```

```
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHIFT** - mode of shift of the price chart from the right border.

```
//+------------------------------------------------------------------+
//| Checks if shifting a price chart from the right border is enabled |
//+------------------------------------------------------------------+
bool ChartShiftGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------------------+
//| Enables/disables displaying of a price chart with a shift from the right border |
//+--------------------------------------------------------------------------------+
bool ChartShiftSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_AUTOSCROLL** - the mode of automatic shift to the right border of the chart.

```
//+------------------------------------------------------------------+
//| Checks if automatic scrolling of a chart to the right            |
//| on new ticks arrival is enabled                                  |
//+------------------------------------------------------------------+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables automatic scrolling of a chart to the right     |
//| on new ticks arrival                                             |
//+------------------------------------------------------------------+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALE** - chart scale property.

```
//+------------------------------------------------------------------+
//| Gets chart scale (from 0 to 5)                                   |
//+------------------------------------------------------------------+
int ChartScaleGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
//+------------------------------------------------------------------+
//| Sets chart scale (from 0 to 5)                                   |
//+------------------------------------------------------------------+
bool ChartScaleSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALEFIX** - the mode of fixed chart scale.

```
//+------------------------------------------------------------------+
//| Checks if the fixed scale mode is enabled                        |
//+------------------------------------------------------------------+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the fixed scale mode                            |
//+------------------------------------------------------------------+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALEFIX_11** - 1:1 chart scale mode.

```
//+------------------------------------------------------------------+
//| Checks if the "1:1" scale is enabled                             |
//+------------------------------------------------------------------+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
```

```
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the "1:1" scale mode                            |
//+------------------------------------------------------------------+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALE_PT_PER_BAR** - the mode of specifying the chart scale in points per bar.

```
//+------------------------------------------------------------------+
//| Checks if the "points per bar" chart scaling mode is enabled     |
//+------------------------------------------------------------------+
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
```

```
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables the "points per bar" chart scaling mode         |
//+------------------------------------------------------------------+
bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_OHLC** - the property of displaying OHLC values in the upper left corner.

```
//+------------------------------------------------------------------------------
//| Checks if displaying of OHLC values in the upper left corner of chart is enabled
//+------------------------------------------------------------------------------
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------------+
//| Enables/disables displaying of OHLC values in the upper left corner of chart |
//+------------------------------------------------------------------------+
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_BID_LINE** – the property of displaying Bid value as a horizontal line on the chart.

```
//+------------------------------------------------------------------+
//| Checks if displaying of Bid line on chart is enabled             |
//+------------------------------------------------------------------+
bool ChartShowBidLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of Bid line on chart                 |
//+------------------------------------------------------------------+
bool ChartShowBidLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_ASK_LINE** – the property of displaying Ask value as a horizontal line on a chart.

```
//+------------------------------------------------------------------+
//| Checks if displaying of Ask line on chart is enabled             |
//+------------------------------------------------------------------+
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of Ask line on chart                 |
//+------------------------------------------------------------------+
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_LAST_LINE** - the property of displaying Last value as a horizontal line on a chart.

```
//+--------------------------------------------------------------------------+
//| Checks if displaying of line for the last performed deal's price is enabled |
//+--------------------------------------------------------------------------+
bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------+
//| Enables/disables displaying of line for the last performed deal's price |
//+--------------------------------------------------------------------+
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_PERIOD_SEP** - the property of displaying vertical separators between adjacent periods.

```
//+----------------------------------------------------------------------------+
//| Checks if displaying of vertical separators between adjacent periods is enabled |
//+------------------------------------------------------------------------------------+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------------------+
//| Enables/disables displaying of vertical separators between adjacent periods |
//+--------------------------------------------------------------------------------+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_GRID** – the property of displaying the chart grid.

```
//+------------------------------------------------------------------+
//| Checks if the chart grid is displayed                            |
//+------------------------------------------------------------------+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of grid on chart                     |
//+------------------------------------------------------------------+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_VOLUMES** – the property of displaying the volumes on a chart.

```
//+------------------------------------------------------------------+
//| Checks if volumes are displayed on a chart                       |
//| The flag indicates the volumes showing mode                      |
//+------------------------------------------------------------------+
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=WRONG_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((ENUM_CHART_VOLUME_MODE)result);
  }
//+------------------------------------------------------------------+
//| Sets mode of displaying volumes on chart                         |
//+------------------------------------------------------------------+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_VOLUMES,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_OBJECT_DESCR** - the property of graphical object pop-up descriptions.

```
//+------------------------------------------------------------------+
//| Checks if pop-up descriptions of graphical objects are displayed  |
//| when hovering mouse over them                                     |
//+------------------------------------------------------------------+
bool ChartShowObjectDescriptionGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------------+
//| Enables/disables displaying of pop-up descriptions of graphical objects |
//| when hovering mouse over them                                           |
//+------------------------------------------------------------------------+
bool ChartShowObjectDescriptionSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_VISIBLE_BARS** defines the number of bars on a chart that are available for display.

```
//+------------------------------------------------------------------+
//| Gets the number of bars that are displayed (visible) in chart window |
//+------------------------------------------------------------------+
int ChartVisibleBars(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOWS_TOTAL** defines the total number of chart windows including indicator subwindows.

```
//+------------------------------------------------------------------+
//| Gets the total number of chart windows including indicator subwindows |
//+------------------------------------------------------------------+
int ChartWindowsTotal(const long chart_ID=0)
  {
```

```
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOW_IS_VISIBLE** defines the subwindow's visibility.

```
//+------------------------------------------------------------------+
//| Checks if the current chart window or subwindow is visible       |
//+------------------------------------------------------------------+
bool ChartWindowsIsVisible(bool &result,const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_IS_VISIBLE,sub_window,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
```

- **CHART_WINDOW_HANDLE** returns the chart handle.

```
//+------------------------------------------------------------------+
//| Gets the chart handle                                            |
//+------------------------------------------------------------------+
int ChartWindowsHandle(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_HANDLE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOW_YDISTANCE** defines the distance in pixels between the upper frame of the indicator subwindow and the upper frame of the chart's main window.

```
//+------------------------------------------------------------------+
//| Gets the distance in pixels between the upper border of          |
//| subwindow and the upper border of chart's main window            |
//+------------------------------------------------------------------+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_FIRST_VISIBLE_BAR** returns the number of the first visible bar on the chart (bar indexing corresponds to the time series).

```
//+------------------------------------------------------------------------+
//| Gets the index of the first visible bar on chart.                      |
//| Indexing is performed like in timeseries: latest bars have smallest indices. |
//+------------------------------------------------------------------------+
int ChartFirstVisibleBar(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
```

```
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_FIRST_VISIBLE_BAR,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WIDTH_IN_BARS** returns the chart width in bars.

```
//+------------------------------------------------------------------+
//| Gets the width of chart (in bars)                                |
//+------------------------------------------------------------------+
int ChartWidthInBars(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WIDTH_IN_PIXELS** returns the chart width in pixels.

```
//+------------------------------------------------------------------+
//| Gets the width of chart (in pixels)                              |
//+------------------------------------------------------------------+
int ChartWidthInPixels(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_HEIGHT_IN_PIXELS** - chart height property in pixels.

```
//+------------------------------------------------------------------+
//| Gets the height of chart (in pixels)                             |
//+------------------------------------------------------------------+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
//+------------------------------------------------------------------+
//| Sets the height of chart (in pixels)                             |
//+------------------------------------------------------------------+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const int sub_windo
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_BACKGROUND -** chart background color.

```
//+------------------------------------------------------------------+
//| Gets the background color of chart                               |
//+------------------------------------------------------------------+
color ChartBackColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive chart background color
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the background color of chart                               |
//+------------------------------------------------------------------+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the chart background color
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_FOREGROUND** - color of axes, scale and OHLC line.

```
//+------------------------------------------------------------------+
//| Gets the color of axes, scale and OHLC line                      |
//+------------------------------------------------------------------+
color ChartForeColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of axes, scale and OHLC line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
```

```
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of axes, scale and OHLC line                      |
//+------------------------------------------------------------------+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of axes, scale and OHLC line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_GRID** - chart grid color.

```
//+------------------------------------------------------------------+
//| Gets the color of chart grid                                     |
//+------------------------------------------------------------------+
color ChartGridColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive chart grid color
   if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of chart grid                                     |
//+------------------------------------------------------------------+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set chart grid color
   if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_VOLUME** - color of volumes and position opening levels.

```
//+------------------------------------------------------------------+
//| Gets the color of volumes and market entry levels                |
//+------------------------------------------------------------------+
color ChartVolumeColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of volumes and market entry levels
   if(!ChartGetInteger(chart_ID,CHART_COLOR_VOLUME,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of volumes and market entry levels                |
//+------------------------------------------------------------------+
bool ChartVolumeColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of volumes and market entry levels
   if(!ChartSetInteger(chart_ID,CHART_COLOR_VOLUME,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
      }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CHART_UP** - color of up bar, its shadow and border of a bullish candlestick's body.

```
//+------------------------------------------------------------------------------+
//| Gets the color of up bar, shadow and border of a bullish candlestick's body |
//+------------------------------------------------------------------------------+
color ChartUpColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of up bar, its shadow and border of bullish candlestick's body
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+-------------------------------------------------------------------+
//| Sets the color of up bar, shadow and border of a bullish candlestick's body |
//+-------------------------------------------------------------------+
bool ChartUpColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of up bar, its shadow and border of body of a bullish candlestick
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CHART_DOWN** - color of down bar, its shadow and border of bearish candlestick's body.

```
//+--------------------------------------------------------------------------------+
//| Gets the color of down bar, shadow and border of a bearish candlestick's body |
//+-------------------------------------------------------------------------------+
color ChartDownColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of down bar, its shadow and border of bearish candlestick's bo
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+-------------------------------------------------------------------------------+
//| Sets the color of down bar, shadow and border of a bearish candlestick's body |
//+-------------------------------------------------------------------------------+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of down bar, its shadow and border of bearish candlestick's body
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CHART_LINE** - color of the chart line and Doji candlesticks.

```
//+------------------------------------------------------------------+
//| Gets the color of chart line and Doji candlesticks               |
//+------------------------------------------------------------------+
color ChartLineColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of the chart line and Doji candlesticks
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
```

```
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of chart line and Doji candlesticks               |
//+------------------------------------------------------------------+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of the chart line and Doji candlesticks
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BULL** - color of bullish candlestick's body.

```
//+------------------------------------------------------------------+
//| Gets the color of bullish candlestick's body                     |
//+------------------------------------------------------------------+
color ChartBullColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of bullish candlestick's body
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of bullish candlestick's body                     |
//+------------------------------------------------------------------+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of bullish candlestick's body
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BEAR** - color of bearish candlestick's body.

```
//+------------------------------------------------------------------+
//| Gets the color of bearish candlestick's body                     |
//+------------------------------------------------------------------+
color ChartBearColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of bearish candlestick's body
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of bearish candlestick's body                     |
//+------------------------------------------------------------------+
bool ChartBearColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of bearish candlestick's body
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_BID** - Bid price line color.

```
//+------------------------------------------------------------------+
//| Gets the color of Bid line                                       |
//+------------------------------------------------------------------+
color ChartBidColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of Bid price line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of Bid line                                       |
//+------------------------------------------------------------------+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of Bid price line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_ASK** - Ask price line color.

```
//+------------------------------------------------------------------+
//| Gets the color of Ask line                                       |
//+------------------------------------------------------------------+
color ChartAskColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of Ask price line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of Ask line                                       |
//+------------------------------------------------------------------+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of Ask price line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_LAST** - color of the last performed deal's price line (Last).

```
//+------------------------------------------------------------------+
//| Gets the color of the last performed deal's price line           |
//+------------------------------------------------------------------+
color ChartLastColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of the last performed deal's price line (Last)
   if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of the last performed deal's price line           |
//+------------------------------------------------------------------+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of the last performed deal's price line (Last)
   if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_STOP_LEVEL** - stop order level color (Stop Loss and Take Profit).

```
//+------------------------------------------------------------------+
//| Gets the color of Stop Loss and Take Profit levels               |
//+------------------------------------------------------------------+
color ChartStopLevelColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of stop order levels (Stop Loss and Take Profit)
   if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| Sets the color of Stop Loss and Take Profit levels               |
//+------------------------------------------------------------------+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
```

```
  {
//--- reset the error value
   ResetLastError();
//--- set the color of stop order levels (Stop Loss and Take Profit)
   if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_TRADE_LEVELS** - property of displaying trade levels on the chart (levels of open positions, Stop Loss, Take Profit and pending orders).

```
//+------------------------------------------------------------------+
//| Checks if trading levels are displayed on chart                  |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of trading levels                    |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_DRAG_TRADE_LEVELS** - property of enabling the ability to drag trading levels on a chart using mouse.

```
//+------------------------------------------------------------------+
//| Checks if dragging of trading levels on chart using mouse is allowed |
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables dragging of trading levels on chart using mouse |
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_DATE_SCALE** - property of displaying the time scale on a chart.

```
//+------------------------------------------------------------------+
//| Checks if the time scale is displayed on chart                   |
//+------------------------------------------------------------------+
bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of the time scale on chart           |
//+------------------------------------------------------------------+
bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_PRICE_SCALE** - property of displaying the price scale on a chart.

```
//+------------------------------------------------------------------+
//| Checks if the price scale is displayed on chart                  |
//+------------------------------------------------------------------+
bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of the price scale on chart          |
//+------------------------------------------------------------------+
bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_ONE_CLICK** - property of displaying the "One click trading" panel on a chart.

```
//+------------------------------------------------------------------+
//| Checks if the "One click trading" panel is displayed on chart    |
//+------------------------------------------------------------------+
bool ChartShowOneClickPanelGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Enables/disables displaying of the "One click trading" panel     |
//| on chart                                                         |
//+------------------------------------------------------------------+
bool ChartShowOneClickPanelSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHIFT_SIZE** - shift size of the zero bar from the right border in percentage values.

```
//+------------------------------------------------------------------+
//| Gets the size of shifting of the zero bar from the right border |
//| of the chart in percentage values (from 10% up to 50%)          |
//+------------------------------------------------------------------+
double ChartShiftSizeGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| Gets the size of shifting of the zero bar from the right border     |
//| of the chart in percentage values (from 10% up to 50%).             |
//| To enable the shift mode, CHART_SHIFT property value should be set to true. |
//+------------------------------------------------------------------+
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FIXED_POSITION** - chart fixed position from the left border in percentage value.

```
//+------------------------------------------------------------------------
//| Gets the location of chart's fixed position from the left border (in percentage va
//+------------------------------------------------------------------------
double ChartFixedPositionGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------------
//| Gets the location of chart's fixed position from the left border (in percentage va
//| To view the location of chart's fixed position, the value of CHART_AUTOSCROLL prop
//| should be set to false.
```

```
//+------------------------------------------------------------------------------
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FIXED_MAX** – property of the chart's fixed maximum.

```
//+------------------------------------------------------------------+
//| Gets the value of chart's fixed maximum                          |
//+------------------------------------------------------------------+
double ChartFixedMaxGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| Sets the value of chart's fixed maximum.                         |
//| To change the value of the property, CHART_SCALEFIX property     |
//| value should be preliminarily set to true.                       |
//+------------------------------------------------------------------+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FIXED_MIN** - property of the chart's fixed minimum.

```
//+------------------------------------------------------------------+
//| Gets the value of chart's fixed minimum                          |
//+------------------------------------------------------------------+
double ChartFixedMinGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| Sets the value of chart's fixed minimum.                         |
//| To change the value of the property, CHART_SCALEFIX property     |
//| value should be preliminarily set to true.                       |
//+------------------------------------------------------------------+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_POINTS_PER_BAR** - value of scale in points per bar.

```
//+------------------------------------------------------------------+
//| Gets the value of chart scale in points per bar                  |
//+------------------------------------------------------------------+
double ChartPointsPerBarGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+----------------------------------------------------------------------+
//| Sets the value of chart scale in points per bar.                     |
//| To view the result of this property's value change, the value of     |
//| CHART_SCALE_PT_PER_BAR property should be preliminarily set to true. |
//+----------------------------------------------------------------------+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_PRICE_MIN** returns the value of the chart minimum.

```
//+----------------------------------------------------------------------+
//| Gets the value of chart minimum in the main window or in a subwindow |
//+----------------------------------------------------------------------+
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
```

```
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
```

- **CHART_PRICE_MAX** returns the value of the chart maximum.

```
//+------------------------------------------------------------------+
//| Gets the value of chart maximum in the main window or in a subwindow |
//+------------------------------------------------------------------+
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
```

- **CHART_COMMENT** - comment on the chart.

```
//+------------------------------------------------------------------+
//| Gets comment in the upper left corner of chart                   |
//+------------------------------------------------------------------+
bool ChartCommentGet(string &result,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetString(chart_ID,CHART_COMMENT,result))
```

```
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Gets comment in the upper left corner of chart                    |
//+------------------------------------------------------------------+
bool ChartCommentSet(const string str,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetString(chart_ID,CHART_COMMENT,str))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_IS_MAXIMIZED** - chart window is maximized.

```
//+------------------------------------------------------------------+
//| Defines if the current chart window is maximized                  |
//+------------------------------------------------------------------+
bool ChartWindowsIsMaximized(bool &result,const long chart_ID=0)
  {
//--- prepare the variable for receiving the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_IS_MAXIMIZED))
     {
      //--- display an error message in the Experts log
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the chart property value in the variable
   result=value;
//--- successful execution
   return(true);
  }
```

- **CHART_IS_MINIMIZED** - chart window is minimized.

```
//+------------------------------------------------------------------+
//| Defines if the current chart window is minimized                 |
//+------------------------------------------------------------------+
bool ChartWindowsIsMinimized(bool &result,const long chart_ID=0)
  {
//--- prepare the variable for receiving the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_IS_MINIMIZED))
     {
      //--- display an error message in the Experts log
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the chart property value in the variable
   result=value;
//--- successful execution
   return(true);
  }
```

Panel for chart properties

```
//--- connect the library of control elements
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- predefined constants
#define X_PROPERTY_NAME_1    10  // x coordinate of the property name in the first co
#define X_PROPERTY_VALUE_1   225 // x coordinate of the property value in the first c
#define X_PROPERTY_NAME_2    345 // x coordinate of the property name in the second a
#define X_PROPERTY_VALUE_2   550 // x coordinate of the property value in the second
#define X_BUTTON_1           285 // x coordinate of the button in the first column
#define X_BUTTON_2           700 // x coordinate of the button in the second column
#define Y_PROPERTY_1         30  // y coordinate of the beginning of the first and se
#define Y_PROPERTY_2         286 // y coordinate of the beginning of the third column
#define Y_DISTANCE           16  // y axial distance between the lines
#define LAST_PROPERTY_NUMBER 111 // number of the last graphical property
//--- input parameters
input color InpFirstColor=clrDodgerBlue; // Color of odd lines
input color InpSecondColor=clrGoldenrod; // Color of even lines
//--- variables and arrays
CChartObjectLabel  ExtLabelsName[];  // labels for displaying property names
CChartObjectLabel  ExtLabelsValue[]; // labels for displaying property values
CChartObjectButton ExtButtons[];     // buttons
int                ExtNumbers[];     // property indices
string             ExtNames[];       // property names
uchar              ExtDataTypes[];   // property data types (integer, double, string)
uint               ExtGroupTypes[];  // array that stores the data on belonging of pr
uchar              ExtDrawTypes[];   // array that stores the data on the type of pro
```

```
double             ExtMaxValue[];    // maximum property values that are possible wher
double             ExtMinValue[];    // minimum property values that are possible wher
double             ExtStep[];        // steps for changing properties
int                ExtCount;         // total number of all properties
color              ExtColors[2];     // array of colors for displaying lines
string             ExtComments[2];   // array of comments (for CHART_COMMENT property)
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- display a comment on the chart
   Comment("SomeComment");
//--- store colors in the array to be able to switch between them later
   ExtColors[0]=InpFirstColor;
   ExtColors[1]=InpSecondColor;
//--- store comments in the array to be able to switch between them later
   ExtComments[0]="FirstComment";
   ExtComments[1]="SecondComment";
//--- prepare and display the control panel for managing chart properties
   if(!PrepareControls())
      return(INIT_FAILED);
//--- successful execution
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Deinitialization function of the expert                          |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- remove the comment on the chart
   Comment("");
  }
//+------------------------------------------------------------------+
//| Handler of a chart event                                         |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- check the event of clicking the chart object
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      //--- divide the object name by separator
      string obj_name[];
      StringSplit(sparam,'_',obj_name);
      //--- check if the object is a button
      if(obj_name[0]=="Button")
```

```
        {
         //--- receive button index
         int index=(int)StringToInteger(obj_name[1]);
         //--- unpress the button
         ExtButtons[index].State(false);
         //--- set the new value of the property depending on its type
         if(ExtDataTypes[index]=='I')
            ChangeIntegerProperty(index);
         if(ExtDataTypes[index]=='D')
            ChangeDoubleProperty(index);
         if(ExtDataTypes[index]=='S')
            ChangeStringProperty(index);
        }
     }
//--- re-draw property values
   RedrawProperties();
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Changes an integer property of chart                             |
//+------------------------------------------------------------------+
void ChangeIntegerProperty(const int index)
  {
//--- receive the current property value
   long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index]);
//--- define the following property value
   switch(ExtDrawTypes[index])
     {
      case 'C':
         value=GetNextColor((color)value);
         break;
      default:
         value=(long)GetNextValue((double)value,index);
         break;
     }
//--- set the new property value
   ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,value);
  }
//+------------------------------------------------------------------+
//| Changes a double property of chart                               |
//+------------------------------------------------------------------+
void ChangeDoubleProperty(const int index)
  {
//--- receive the current property value
   double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index]);
//--- define the following property value
   value=GetNextValue(value,index);
//--- set the new property value
   ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
  }
//+------------------------------------------------------------------+
//| Changes a string property of chart                               |
//+------------------------------------------------------------------+
void ChangeStringProperty(const int index)
  {
//--- static variable for switching inside ExtComments array
   static uint comment_index=1;
//--- change index for receiving another comment
   comment_index=1-comment_index;
//--- set the new property value
   ChartSetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComments[comment_
  }
//+------------------------------------------------------------------+
//| Gets the next property value                                     |
//+------------------------------------------------------------------+
double GetNextValue(const double value,const int index)
  {
   if(value+ExtStep[index]<=ExtMaxValue[index])
      return(value+ExtStep[index]);
   else
      return(ExtMinValue[index]);
  }
//+------------------------------------------------------------------+
//| Gets the next color for color type property                      |
//+------------------------------------------------------------------+
color GetNextColor(const color clr)
  {
//--- return the following color value
   switch(clr)
     {
      case clrWhite: return(clrRed);
      case clrRed:   return(clrGreen);
      case clrGreen: return(clrBlue);
      case clrBlue:  return(clrBlack);
      default:       return(clrWhite);
     }
  }
//+------------------------------------------------------------------+
//| Re-draws property values                                         |
//+------------------------------------------------------------------+
void RedrawProperties(void)
  {
//--- property value text
   string text;
   long   value;
//--- loop of the number of properties
   for(int i=0;i<ExtCount;i++)
     {
      text="";
      switch(ExtDataTypes[i])
        {
         case 'I':
            //--- receive the current property value
            if(!ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[i],0,value))
            break;
            //--- integer property text
            switch(ExtDrawTypes[i])
              {
               //--- color property
               case 'C':
                  text=(string)((color)value);
                  break;
                  //--- boolean property
               case 'B':
                  text=(string)((bool)value);
                  break;
                  //--- ENUM_CHART_MODE enumeration property
               case 'M':
                  text=EnumToString((ENUM_CHART_MODE)value);
                  break;
                  //--- ENUM_CHART_VOLUME_MODE enumeration property
               case 'V':
                  text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                  break;
                  //--- int type number
               default:
                  text=IntegerToString(value);
                  break;
              }
            break;
         case 'D':
            //--- double property text
            text=DoubleToString(ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumber
            break;
         case 'S':
            //--- string property text
            text=ChartGetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[i]);
            break;
        }
      //--- display property value
      ExtLabelsValue[i].Description(text);
     }
  }
//+------------------------------------------------------------------+
//| Creates panel for managing chart properties                      |
//+------------------------------------------------------------------+
bool PrepareControls(void)
```

```
  {
//--- allocate memory for arrays with a reserve
   MemoryAllocation(LAST_PROPERTY_NUMBER+1);
//--- variables
   int i=0;     // loop variable
   int col_1=0; // number of properties in the first column
   int col_2=0; // number of properties in the second column
   int col_3=0; // number of properties in the third column
//--- current number of properties - 0
   ExtCount=0;
//--- looking for properties in the loop
   while(i<=LAST_PROPERTY_NUMBER)
     {
      //--- store the current number of the property
      ExtNumbers[ExtCount]=i;
      //--- increase the value of the loop variable
      i++;
      //--- check if there is a property with such a number
      if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[ExtCount],E
        {
         //--- create control elements for the property
         switch(ExtGroupTypes[ExtCount])
           {
            case 1:
               //--- create labels and a button for the property
               if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VALUE_1,X_BUT
               return(false);
               //--- number of the elements in the first column has increased
               col_1++;
               break;
            case 2:
               //--- create labels and a button for the property
               if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,X_BUT
               return(false);
               //--- number of the elements in the second column has increased
               col_2++;
               break;
            case 3:
               //--- create only labels for the property
               if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,0,Y_P
               return(false);
               //--- number of the elements in the third column has increased
               col_3++;
               break;
           }
         //--- define maximum and minimum property value and step
         GetMaxMinStep(ExtNumbers[ExtCount],ExtMaxValue[ExtCount],ExtMinValue[ExtCount
         //--- increase the number of properties
         ExtCount++;
        }
     }
//--- free the memory not used by arrays
   MemoryAllocation(ExtCount);
//--- re-draw property values
   RedrawProperties();
   ChartRedraw();
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Allocates memory for arrays                                      |
//+------------------------------------------------------------------+
void MemoryAllocation(const int size)
  {
   ArrayResize(ExtLabelsName,size);
   ArrayResize(ExtLabelsValue,size);
   ArrayResize(ExtButtons,size);
   ArrayResize(ExtNumbers,size);
   ArrayResize(ExtNames,size);
   ArrayResize(ExtDataTypes,size);
   ArrayResize(ExtGroupTypes,size);
   ArrayResize(ExtDrawTypes,size);
   ArrayResize(ExtMaxValue,size);
   ArrayResize(ExtMinValue,size);
   ArrayResize(ExtStep,size);
  }
//+------------------------------------------------------------------+
//| Checks if the property index belongs to the one of               |
//| ENUM_CHART_PROPERTIES enumerations                               |
//+------------------------------------------------------------------+
bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_type,uchar &
  {
//--- check if the property is of integer type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
   if(_LastError==0)
     {
      data_type='I';                      // property from ENUM_CHART_PROPERTY_INTEGE
      GetTypes(ind,group_type,draw_type); // define property display parameters
      return(true);
     }
//--- check if the property is of double type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
   if(_LastError==0)
     {
      data_type='D';                      // property from ENUM_CHART_PROPERTY_DOUBLE
      GetTypes(ind,group_type,draw_type); // define property display parameters
      return(true);
     }
//--- check if the property is of string type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
   if(_LastError==0)
     {
      data_type='S';                          // property from ENUM_CHART_PROPERTY_STRING
      GetTypes(ind,group_type,draw_type); // define property display parameters
      return(true);
     }
//--- property does not belong to any enumeration
   return(false);
  }
//+------------------------------------------------------------------+
//| Defines the group in which property should be stored,            |
//| as well as its display type                                      |
//+------------------------------------------------------------------+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- check if the property belongs to the third group
//--- third group properties are displayed in the second column starting from CHART_BI
   if(CheckThirdGroup(property_number,group_type,draw_type))
      return;
//--- check if the property belongs to the second group
//--- second group properties are displayed at the beginning of the second column
   if(CheckSecondGroup(property_number,group_type,draw_type))
      return;
//--- if you find yourself here, the property belongs to the first group (first column
   CheckFirstGroup(property_number,group_type,draw_type);
  }
//+----------------------------------------------------------------------+
//| Checks if property belongs to the third group and                    |
//| defines its display type in case of a positive answer                |
//+----------------------------------------------------------------------+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- check if the property belongs to the third group
   switch(property_number)
     {
      //--- boolean properties
      case CHART_IS_OBJECT:
      case CHART_WINDOW_IS_VISIBLE:
         draw_type='B';
         break;
         //--- integer properties
      case CHART_VISIBLE_BARS:
      case CHART_WINDOWS_TOTAL:
      case CHART_WINDOW_HANDLE:
```

```
      case CHART_WINDOW_YDISTANCE:
      case CHART_FIRST_VISIBLE_BAR:
      case CHART_WIDTH_IN_BARS:
      case CHART_WIDTH_IN_PIXELS:
         draw_type='I';
         break;
         //--- double properties
      case CHART_PRICE_MIN:
      case CHART_PRICE_MAX:
         draw_type='D';
         break;
         //--- in fact, this property is a command of displaying the chart on top of
         //--- there is no need to apply this panel, as the window will always be
         //--- on top of other ones before we use it
      case CHART_BRING_TO_TOP:
         draw_type=' ';
         break;
         //--- property does not belong to the third group
      default:
         return(false);
     }
//--- property belongs to the third group
   group_type=3;
   return(true);
  }
//+------------------------------------------------------------------+
//| Checks if property belongs to the second group and               |
//| defines its display type in case of a positive answer            |
//+------------------------------------------------------------------+
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- check if the property belongs to the second group
   switch(property_number)
     {
      //--- ENUM_CHART_MODE type property
      case CHART_MODE:
         draw_type='M';
         break;
         //--- ENUM_CHART_VOLUME_MODE type property
      case CHART_SHOW_VOLUMES:
         draw_type='V';
         break;
         //--- string property
      case CHART_COMMENT:
         draw_type='S';
         break;
         //--- color property
      case CHART_COLOR_BACKGROUND:
      case CHART_COLOR_FOREGROUND:
         case CHART_COLOR_GRID:
         case CHART_COLOR_VOLUME:
         case CHART_COLOR_CHART_UP:
         case CHART_COLOR_CHART_DOWN:
         case CHART_COLOR_CHART_LINE:
         case CHART_COLOR_CANDLE_BULL:
         case CHART_COLOR_CANDLE_BEAR:
         case CHART_COLOR_BID:
         case CHART_COLOR_ASK:
         case CHART_COLOR_LAST:
         case CHART_COLOR_STOP_LEVEL:
            draw_type='C';
            break;
            //--- property does not belong to the second group
         default:
            return(false);
        }
//--- property belongs to the second group
   group_type=2;
   return(true);
  }
//+------------------------------------------------------------------+
//| Called only if it is already known that property does not belong |
//| to the second and third property groups                          |
//+------------------------------------------------------------------+
void CheckFirstGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- the property belongs to the first group
   group_type=1;
//--- define property display type
   switch(property_number)
     {
      //--- integer properties
      case CHART_SCALE:
      case CHART_HEIGHT_IN_PIXELS:
         draw_type='I';
         return;
         //--- double properties
      case CHART_SHIFT_SIZE:
      case CHART_FIXED_POSITION:
      case CHART_FIXED_MAX:
      case CHART_FIXED_MIN:
      case CHART_POINTS_PER_BAR:
         draw_type='D';
         return;
         //--- only boolean properties have remained
      default:
         draw_type='B';
         return;
     }
  }
//+------------------------------------------------------------------+
//| Creates label and button for property                            |
//+------------------------------------------------------------------+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
  {
//--- static array for switching inside ExtColors color array
   static uint color_index[3]={1,1,1};
//--- change index for receiving another color
   color_index[type]=1-color_index[type];
//--- display labels and a button (if btn=true) for the property
   if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],ExtColors[colo
      return(false);
   if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[color_index[t
      return(false);
   if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates label                                                    |
//+------------------------------------------------------------------+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string text,
                 const color clr,const int x,const int y)
  {
   if(!lbl.Create(0,"Label_"+name,0,x,y)) return(false);
   if(!lbl.Description(text))             return(false);
   if(!lbl.FontSize(10))                  return(false);
   if(!lbl.Color(clr))                    return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates button                                                   |
//+------------------------------------------------------------------+
bool ButtonCreate(CChartObjectButton &btn,const string name,
                  const int x,const int y)
  {
   if(!btn.Create(0,"Button_"+name,0,x,y,50,15)) return(false);
   if(!btn.Description("Next"))                  return(false);
   if(!btn.FontSize(10))                         return(false);
   if(!btn.Color(clrBlack))                      return(false);
   if(!btn.BackColor(clrWhite))                  return(false);
   if(!btn.BorderColor(clrBlack))                return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Defines maximum and minimum property value and step              |
//+------------------------------------------------------------------+
void GetMaxMinStep(const int property_number,double &max,double &min,double &step)
  {
   double value;
//--- set values depending on the property type
   switch(property_number)
     {
      case CHART_SCALE:
         max=5;
         min=0;
         step=1;
         break;
      case CHART_MODE:
      case CHART_SHOW_VOLUMES:
         max=2;
         min=0;
         step=1;
         break;
      case CHART_SHIFT_SIZE:
         max=50;
         min=10;
         step=2.5;
         break;
      case CHART_FIXED_POSITION:
         max=90;
         min=0;
         step=15;
         break;
      case CHART_POINTS_PER_BAR:
         max=19;
         min=1;
         step=3;
         break;
      case CHART_FIXED_MAX:
         value=ChartGetDouble(0,CHART_FIXED_MAX);
         max=value*1.25;
         min=value;
         step=value/32;
         break;
      case CHART_FIXED_MIN:
         value=ChartGetDouble(0,CHART_FIXED_MIN);
         max=value;
         min=value*0.75;
         step=value/32;
         break;
      case CHART_HEIGHT_IN_PIXELS:
         max=700;
         min=520;
         step=30;
         break;
         //--- default values
      default:
         max=1;
         min=0;
         step=1;
     }
  }
```

Object Constants

There are 44 graphical objects that can be created and displayed in the price chart. All constants for working with objects are divided into 9 groups:

- Object types – Identifiers of graphical objects;
- Object properties – setting and getting properties of graphical objects;
- Methods of object binding – constants of object positioning in the chart;
- Binding corner – setting the corner relative to which an object is positioned on chart;
- Visibility of objects – setting timeframes in which an object is visible;
- Levels of Elliott Waves – gradation of waves;
- Gann objects – trend constants for Gann fan and Gann grid;
- Web colors – constants of predefined web colors;
- Wingdings – codes of characters of the Wingdings font.

Object Types

When a graphical object is created using the ObjectCreate() function, it's necessary to specify the type of object being created, which can be one of the values of the ENUM_OBJECT enumeration. Further specifications of object properties are possible using functions for working with graphical objects.

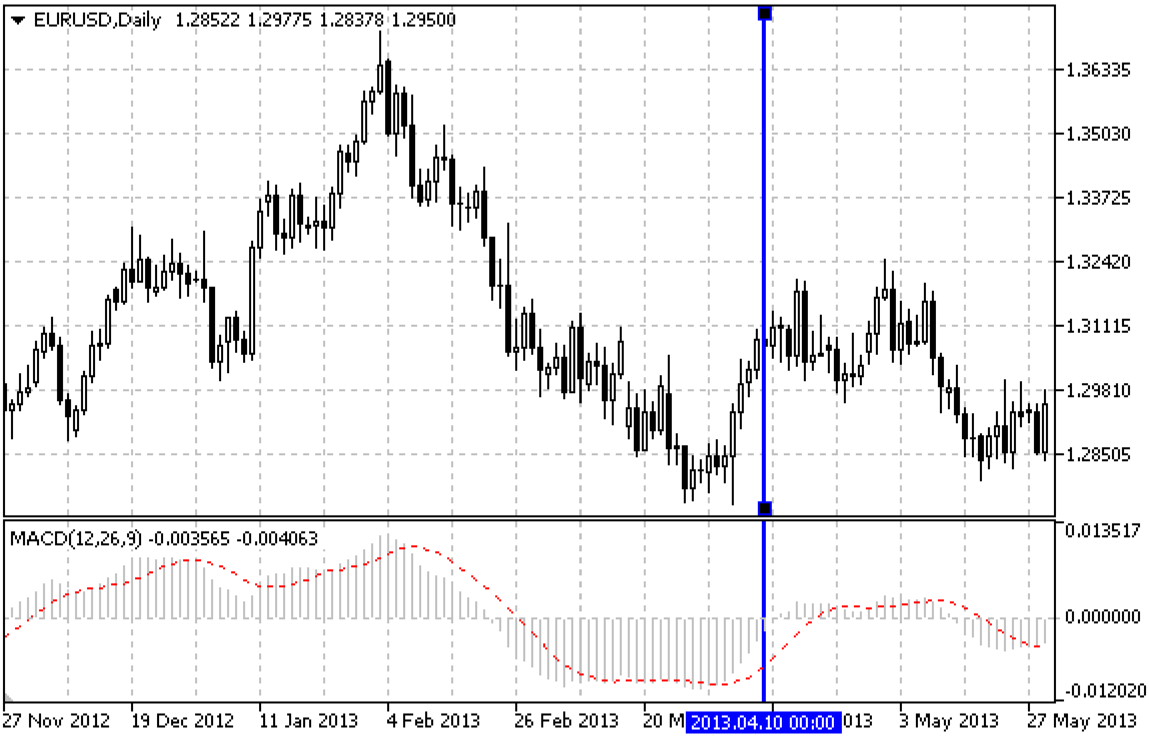
ENUM_OBJECT

ID		Description
OBJ_VLINE	|	Vertical Line
OBJ_HLINE	—	Horizontal Line
OBJ_TREND	/	Trend Line
OBJ_TRENDBYANGLE	∠	Trend Line By Angle
OBJ_CYCLES	⋮	Cycle Lines
OBJ_ARROWED_LINE	↗	Arrowed Line
OBJ_CHANNEL	E	Equidistant Channel
OBJ_STDDEVCHANNEL	D	Standard Deviation Channel
OBJ_REGRESSION	⋰	Linear Regression Channel
OBJ_PITCHFORK	⫽	Andrews’ Pitchfork
OBJ_GANNLINE	G	Gann Line
OBJ_GANNFAN	G	Gann Fan
OBJ_GANNGRID	G	Gann Grid
OBJ_FIBO	F	Fibonacci Retracement
OBJ_FIBOTIMES	F	Fibonacci Time Zones
OBJ_FIBOFAN	F	Fibonacci Fan
OBJ_FIBOARC	F	Fibonacci Arcs
OBJ_FIBOCHANNEL	F	Fibonacci Channel
OBJ_EXPANSION	F	Fibonacci Expansion
OBJ_ELLIOTWAVE5	E	Elliott Motive Wave
OBJ_ELLIOTWAVE3	F	Elliott Correction Wave
OBJ_RECTANGLE	■	Rectangle
OBJ_TRIANGLE	▲	Triangle
OBJ_ELLIPSE	●	Ellipse
OBJ_ARROW_THUMB_UP	👍	Thumbs Up
OBJ_ARROW_THUMB_DOWN	👎	Thumbs Down

ID		Description
OBJ_ARROW_UP	⇧	Arrow Up
OBJ_ARROW_DOWN	⇩	Arrow Down
OBJ_ARROW_STOP	⛔	Stop Sign
OBJ_ARROW_CHECK	✓	Check Sign
OBJ_ARROW_LEFT_PRICE	▭	Left Price Label
OBJ_ARROW_RIGHT_PRICE	▭	Right Price Label
OBJ_ARROW_BUY	⇧	Buy Sign
OBJ_ARROW_SELL	⇩	Sell Sign
OBJ_ARROW	✧	Arrow
OBJ_TEXT	T	Text
OBJ_LABEL	A	Label
OBJ_BUTTON	OK	Button
OBJ_CHART	▣	Chart
OBJ_BITMAP	▣	Bitmap
OBJ_BITMAP_LABEL	▣	Bitmap Label
OBJ_EDIT	I	Edit
OBJ_EVENT	13:30	The "Event" object corresponding to an event in the economic calendar
OBJ_RECTANGLE_LABEL	🟦	The "Rectangle label" object for creating and designing the custom graphical interface.

OBJ_VLINE

Vertical Line.


EURUSD,Daily 1.28522 1.29775 1.28378 1.29500
1.36335
1.35030
1.33725
1.32420
1.31115
1.29810
1.28505
MACD(12,26,9) -0.003565 -0.004063
0.013517
0.000000
-0.012020
27 Nov 2012
19 Dec 2012
11 Jan 2013
4 Feb 2013
26 Feb 2013
20 M
2013.04.10 00:00
013
3 May 2013
27 May 2013

Note

When drawing a vertical line, it is possible to set the line display mode for all chart windows (property OBJPROP_RAY).

Example

The following script creates and moves the vertical line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Vertical Line\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="VLine";      // Line name
input int             InpDate=25;           // Event date, %
input color           InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int             InpWidth=3;           // Line width
input bool            InpBack=false;        // Background line
input bool            InpSelection=true;    // Highlight to move
input bool            InpRay=true;          // Line's continuation down
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the vertical line                                         |
//+------------------------------------------------------------------+
bool VLineCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="VLine",      // line name
                 const int             sub_window=0,      // subwindow index
                 datetime              time=0,            // line time
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the background
                 const bool            selection=true,    // highlight to move
                 const bool            ray=true,          // line's continuation down
                 const bool            hidden=true,       // hidden in the object list
                 const long            z_order=0)         // priority for mouse click
  {
//--- if the line time is not set, draw it via the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- create a vertical line
   if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create a vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of displaying the line in the chart su
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
```

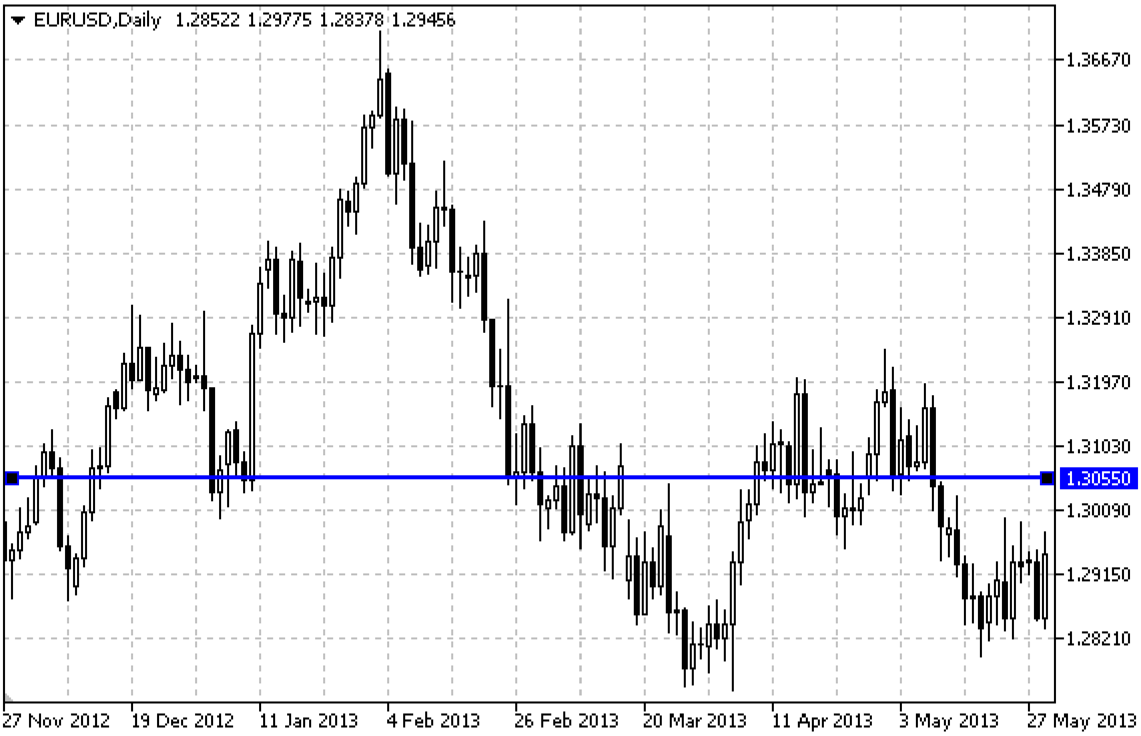
```
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the vertical line                                           |
//+------------------------------------------------------------------+
bool VLineMove(const long   chart_ID=0,   // chart's ID
               const string name="VLine", // line name
               datetime     time=0)       // line time
  {
//--- if line time is not set, move the line to the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the vertical line
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the vertical line                                         |
//+------------------------------------------------------------------+
bool VLineDelete(const long   chart_ID=0,   // chart's ID
                 const string name="VLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete the vertical line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the vertical line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing line anchor point's coordinates
   datetime date[];
//--- memory allocation
   ArrayResize(date,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- define points for drawing the line
   int d=InpDate*(bars-1)/100;
//--- create a vertical line
   if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpRay,InpHidden,InpZOrder))
      return;
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line
//--- loop counter
   int h_steps=bars/2;
//--- move the line
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!VLineMove(0,InpName,date[d]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.03 seconds of delay
      Sleep(30);
     }
```

```
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   VLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_HLINE

Horizontal Line.



Example

The following script creates and moves the horizontal line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

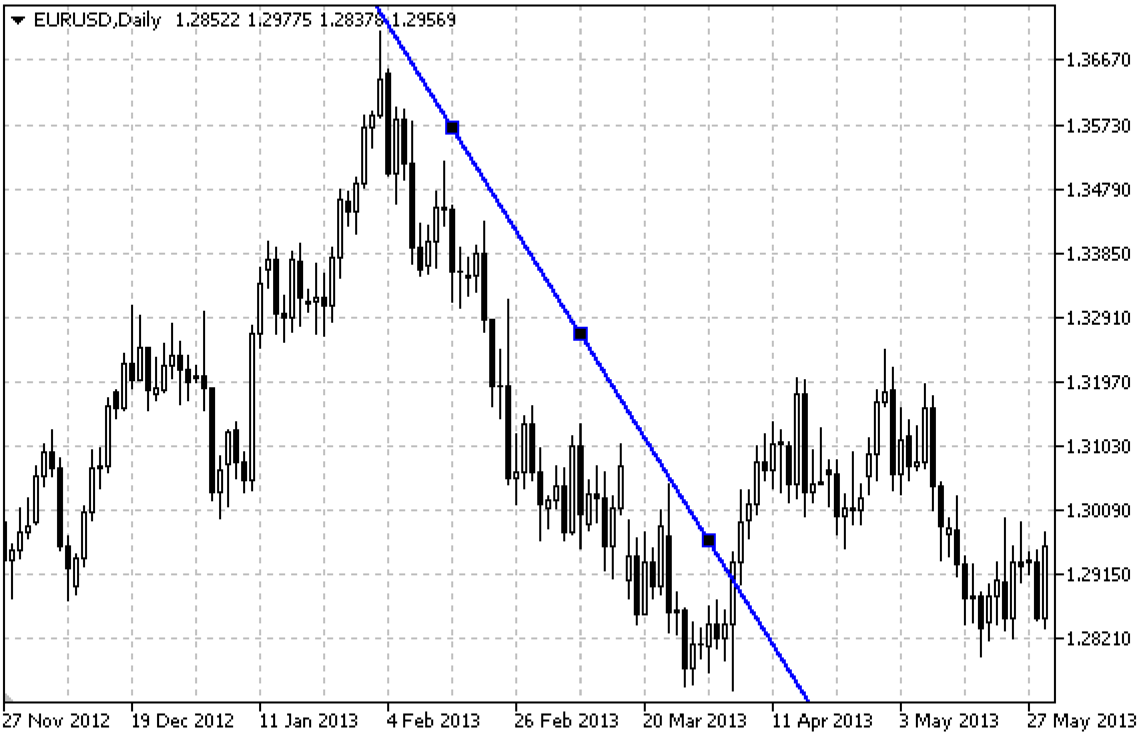
```
//--- description
#property description "Script draws \"Horizontal Line\" graphical object."
#property description "Anchor point price is set in percentage of the height of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="HLine";     // Line name
input int             InpPrice=25;         // Line price, %
input color           InpColor=clrRed;     // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int             InpWidth=3;          // Line width
input bool            InpBack=false;       // Background line
input bool            InpSelection=true;   // Highlight to move
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the horizontal line                                       |
//+------------------------------------------------------------------+
bool HLineCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="HLine",      // line name
                 const int             sub_window=0,      // subwindow index
                 double                price=0,           // line price
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the background
                 const bool            selection=true,    // highlight to move
                 const bool            hidden=true,       // hidden in the object list
                 const long            z_order=0)         // priority for mouse click
  {
//--- if the price is not set, set it at the current Bid price level
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- create a horizontal line
   if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
     {
      Print(__FUNCTION__,
            ": failed to create a horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move horizontal line                                             |
//+------------------------------------------------------------------+
bool HLineMove(const long   chart_ID=0,   // chart's ID
                const string name="HLine", // line name
                double       price=0)      // line price
  {
//--- if the line price is not set, move it to the current Bid price level
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move a horizontal line
   if(!ObjectMove(chart_ID,name,0,0,price))
     {
      Print(__FUNCTION__,
            ": failed to move the horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a horizontal line                                         |
//+------------------------------------------------------------------+
bool HLineDelete(const long   chart_ID=0,   // chart's ID
                 const string name="HLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete a horizontal line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a horizontal line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- price array size
   int accuracy=1000;
```

```
//--- array for storing the price values to be used
//--- for setting and changing line anchor point's coordinates
   double price[];
//--- memory allocation
   ArrayResize(price,accuracy);
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int p=InpPrice*(accuracy-1)/100;
//--- create a horizontal line
   if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line
//--- loop counter
   int v_steps=accuracy/2;
//--- move the line
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!HLineMove(0,InpName,price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete from the chart
   HLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
```

```
//---
  }
```

OBJ_TREND

Trend Line.


EURUSD,Daily 1.28522 1.29775 1.28378 1.29569
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
27 Nov 2012
19 Dec 2012
11 Jan 2013
4 Feb 2013
26 Feb 2013
20 Mar 2013
11 Apr 2013
3 May 2013
27 May 2013

Note

For Trend Line, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Trend Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Trend";     // Line name
input int             InpDate1=35;         // 1 st point's date, %
input int             InpPrice1=60;        // 1 st point's price, %
input int             InpDate2=65;         // 2 nd point's date, %
input int             InpPrice2=40;        // 2 nd point's price, %
input color           InpColor=clrRed;     // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int             InpWidth=2;          // Line width
input bool            InpBack=false;       // Background line
input bool            InpSelection=true;   // Highlight to move
input bool            InpRayLeft=false;    // Line's continuation to the left
input bool            InpRayRight=false;   // Line's continuation to the right
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create a trend line by the given coordinates                     |
//+------------------------------------------------------------------+
bool TrendCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="TrendLine",  // line name
                 const int             sub_window=0,      // subwindow index
                 datetime              time1=0,           // first point time
                 double                price1=0,          // first point price
                 datetime              time2=0,           // second point time
                 double                price2=0,          // second point price
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the background
                 const bool            selection=true,    // highlight to move
                 const bool            ray_left=false,    // line's continuation to th
                 const bool            ray_right=false,   // line's continuation to th
                 const bool            hidden=true,       // hidden in the object list
                 const long            z_order=0)         // priority for mouse click
  {
//--- set anchor points' coordinates if they are not set
   ChangeTrendEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a trend line by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
```

```
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move trend line anchor point                                     |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // line name
                      const int    point_index=0,    // anchor point index
                      datetime     time=0,           // anchor point time coordinate
                      double       price=0)          // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move trend line's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function deletes the trend line from the chart.              |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // chart's ID
                 const string name="TrendLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete a trend line
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of trend line's anchor points and set default   |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

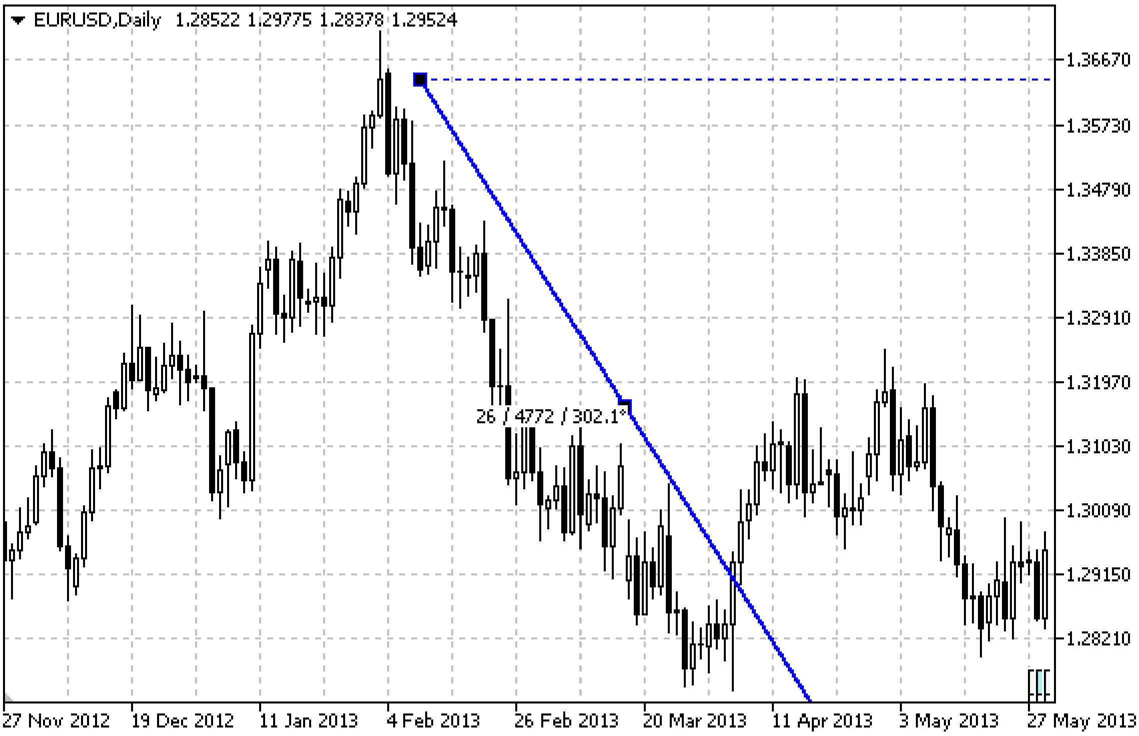
```
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
   if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,InpStyle
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line's anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
```

```
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- move the second anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2<accuracy-1)
         p2+=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- half a second of delay
   Sleep(500);
//--- loop counter
   int h_steps=bars/2;
//--- move both anchor points horizontally at the same time
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- shift the points
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.03 seconds of delay
      Sleep(30);
     }
//--- 1 second of delay
   Sleep(1000);
```

```
//--- delete a trend line
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TRENDBYANGLE

Trend Line By Angle.


EURUSD,Daily 1.28522 1.29775 1.28378 1.29524
26 / 4772 / 302.1°
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
27 Nov 2012
19 Dec 2012
11 Jan 2013
4 Feb 2013
26 Feb 2013
20 Mar 2013
11 Apr 2013
3 May 2013
27 May 2013

Note

For Trend Line By Angle, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

Both angle and the second anchor point's coordinates can be used to set the slope of the line.

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

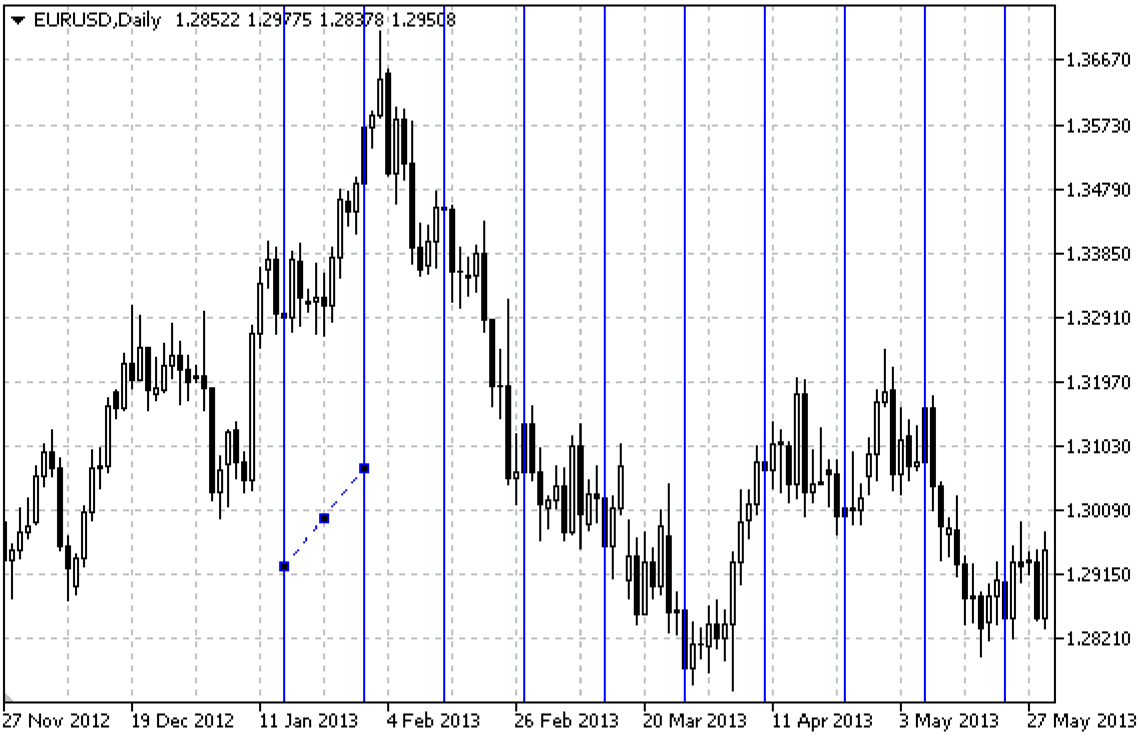
```
//--- description
#property description "Script draws \"Trend Line By Angle\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Trend";      // Line name
input int             InpDate1=50;          // 1 st point's date, %
input int             InpPrice1=75;         // 1 st point's price, %
input int             InpAngle=0;           // Line's slope angle
input color           InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;  // Line style
input int             InpWidth=2;          // Line width
input bool            InpBack=false;       // Background line
input bool            InpSelection=true;   // Highlight to move
input bool            InpRayLeft=false;    // Line's continuation to the left
input bool            InpRayRight=true;    // Line's continuation to the right
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create a trend line by angle                                     |
//+------------------------------------------------------------------+
bool TrendByAngleCreate(const long            chart_ID=0,        // chart's ID
                        const string          name="TrendLine",  // line name
                        const int             sub_window=0,      // subwindow index
                        datetime              time=0,            // point time
                        double                price=0,           // point price
                        const double          angle=45.0,        // slope angle
                        const color           clr=clrRed,        // line color
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                        const int             width=1,           // line width
                        const bool            back=false,        // in the background
                        const bool            selection=true,    // highlight to move
                        const bool            ray_left=false,    // line's continuati
                        const bool            ray_right=true,    // line's continuati
                        const bool            hidden=true,       // hidden in the obje
                        const long            z_order=0)         // priority for mouse
  {
//--- create the second point to facilitate dragging the trend line by mouse
   datetime time2=0;
   double   price2=0;
//--- set anchor points' coordinates if they are not set
   ChangeTrendEmptyPoints(time,price,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a trend line using 2 points
   if(!ObjectCreate(chart_ID,name,OBJ_TRENDBYANGLE,sub_window,time,price,time2,price2)
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- change trend line's slope angle; when changing the angle, coordinates of the se
//--- point of the line are redefined automatically according to the angle's new valu
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change trend line anchor point's coordinates                     |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // line name
                      datetime     time=0,           // anchor point time coordinate
                      double       price=0)          // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move trend line's anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change trend line's slope angle                                  |
//+------------------------------------------------------------------+
bool TrendAngleChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // trend line name
                      const double angle=45)         // trend line's slope angle
  {
//--- reset the error value
   ResetLastError();
//--- change trend line's slope angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change the line's slope angle! Error code = ",GetLastError())
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the trend line                                            |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // chart's ID
                 const string name="TrendLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete a trend line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of trend line's anchor points and set default   |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- set coordinates of the second, auxiliary point
//--- the second point will be 9 bars left and have the same price
   datetime second_point_time[10];
   CopyTime(Symbol(),Period(),time1,10,second_point_time);
   time2=second_point_time[0];
   price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int d1=InpDate1*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create a trend line
   if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor,InpStyle,
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
```

```
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move and rotate the line
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point and change the line's slope angle
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,date[d1],price[p1]))
         return;
      if(!TrendAngleChange(0,InpName,18*(i+1)))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete from the chart
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_CYCLES

Cycle Lines.



Note

The distance between the lines is set by time coordinates of two anchor points of the object.

Example

The following script creates and moves cycle lines on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates cycle lines on the chart."
#property description "Anchor point coordinates are set in percentage"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Cycles";    // Object name
input int             InpDate1=10;         // 1 st point's date, %
input int             InpPrice1=45;        // 1 st point's price, %
input int             InpDate2=20;         // 2 nd point's date, %
input int             InpPrice2=55;        // 2 nd point's price, %
input color           InpColor=clrRed;     // Color of cycle lines
input ENUM_LINE_STYLE InpStyle=STYLE_DOT;  // Style of cycle lines
input int             InpWidth=1;          // Width of cycle lines
input bool            InpBack=false;      // Background object
input bool            InpSelection=true;  // Highlight to move
input bool            InpHidden=true;     // Hidden in the object list
input long            InpZOrder=0;        // Priority for mouse click
//+------------------------------------------------------------------+
//| Create cycle lines                                               |
//+------------------------------------------------------------------+
bool CyclesCreate(const long            chart_ID=0,        // chart's ID
                  const string          name="Cycles",     // object name
                  const int             sub_window=0,      // subwindow index
                  datetime              time1=0,           // first point time
                  double                price1=0,          // first point price
                  datetime              time2=0,           // second point time
                  double                price2=0,          // second point price
                  const color           clr=clrRed,        // color of cycle lines
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // style of cycle lines
                  const int             width=1,           // width of cycle lines
                  const bool            back=false,        // in the background
                  const bool            selection=true,    // highlight to move
                  const bool            hidden=true,       // hidden in the object lis
                  const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor points' coordinates if they are not set
   ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create cycle lines by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create cycle lines! Error code = ",GetLastError());
      return(false);
     }
//--- set color of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the lines by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool CyclesPointChange(const long   chart_ID=0,     // chart's ID
                       const string name="Cycles", // object name
                       const int    point_index=0, // anchor point index
                       datetime     time=0,         // anchor point time coordinate
                       double       price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the cycle lines                                           |
//+------------------------------------------------------------------+
bool CyclesDelete(const long   chart_ID=0,     // chart's ID
                  const string name="Cycles") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete cycle lines
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete cycle lines! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

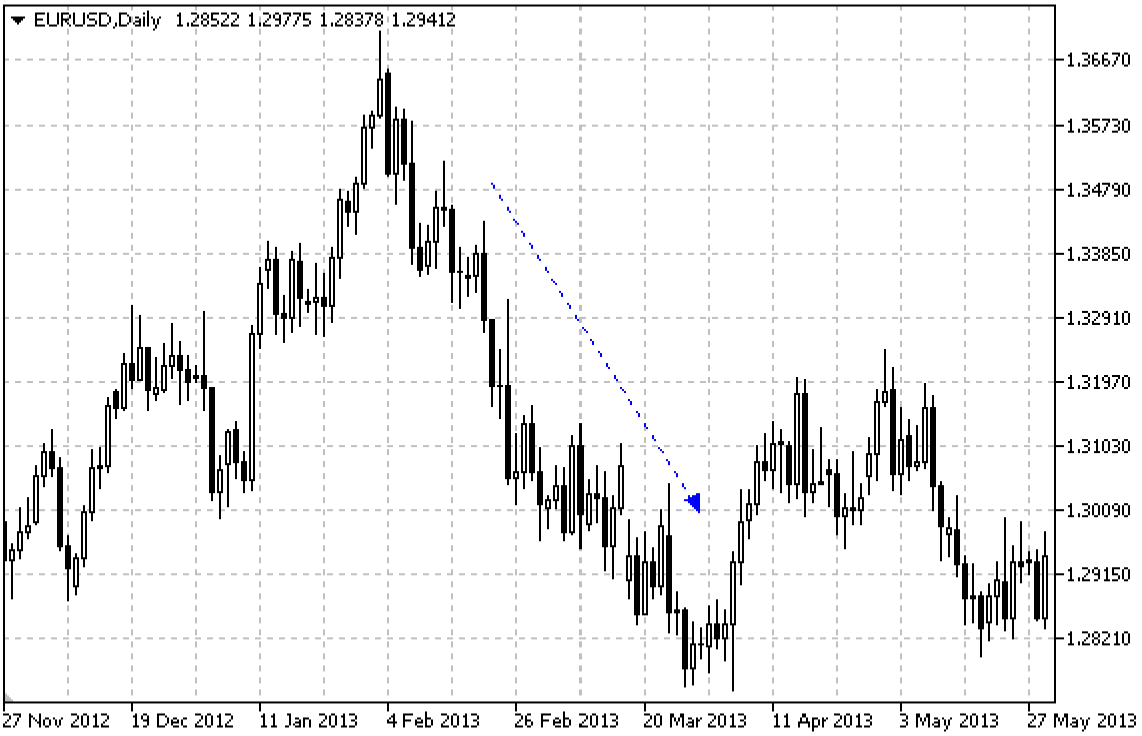
```
//+------------------------------------------------------------------+
//| Check the values of cycle lines' anchor points and set default   |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of cycle lines' anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
```

```
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing cycle lines
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
   if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int h_steps=bars/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
```

```
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   h_steps=bars/4;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1<bars-1)
         d1+=1;
      //--- move the point
      if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   CyclesDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROWED_LINE

Arrowed line.



Example

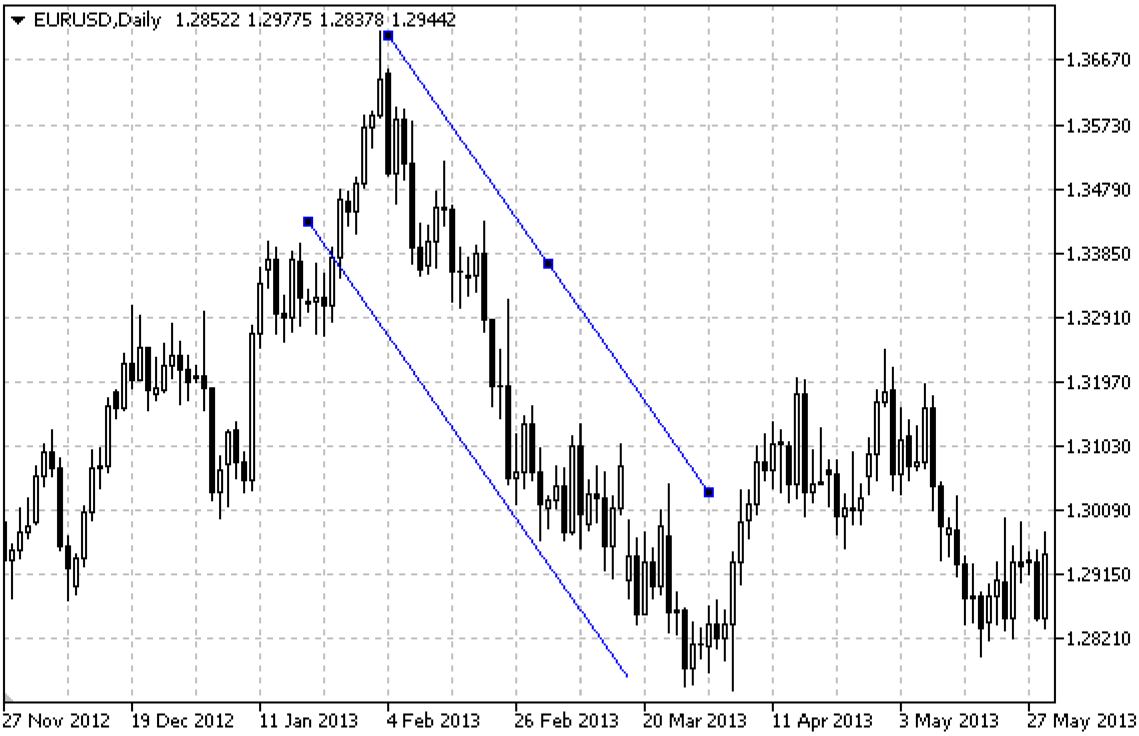
The following script creates and moves an arrow line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Arrowed line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="ArrowedLine"; // Line name
input int             InpDate1=35;           // 1 st point's date, %
input int             InpPrice1=60;          // 1 st point's price, %
input int             InpDate2=65;           // 2 nd point's date, %
input int             InpPrice2=40;          // 2 nd point's price, %
input color           InpColor=clrRed;       // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;   // Line style
input int             InpWidth=2;            // Line width
input bool            InpBack=false;         // Background line
input bool            InpSelection=true;     // Highlight to move
input bool            InpHidden=true;        // Hidden in the object list
input long            InpZOrder=0;           // Priority for mouse click
```

```
//+------------------------------------------------------------------+
//| Create an arrowed line by the given coordinates                  |
//+------------------------------------------------------------------+
bool ArrowedLineCreate(const long            chart_ID=0,         // chart's ID
                       const string          name="ArrowedLine", // line name
                       const int             sub_window=0,       // subwindow index
                       datetime              time1=0,            // first point time
                       double                price1=0,           // first point price
                       datetime              time2=0,            // second point time
                       double                price2=0,           // second point price
                       const color           clr=clrRed,         // line color
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // line style
                       const int             width=1,            // line width
                       const bool            back=false,         // in the background
                       const bool            selection=true,     // highlight to move
                       const bool            hidden=true,        // hidden in the obje
                       const long            z_order=0)          // priority for mouse
  {
//--- set anchor points' coordinates if they are not set
   ChangeArrowedLineEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create an arrowed line by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_ARROWED_LINE,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": failed to create an arrowed line! Error code = ",GetLastError());
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move arrowed line's anchor point                                 |
//+------------------------------------------------------------------+
bool ArrowedLinePointChange(const long   chart_ID=0,         // chart's ID
                            const string name="ArrowedLine", // line name
                            const int    point_index=0,      // anchor point index
                            datetime     time=0,             // anchor point time coo
                            double       price=0)            // anchor point price co
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the line's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes the arrowed line from the chart             |
//+------------------------------------------------------------------+
bool ArrowedLineDelete(const long   chart_ID=0,         // chart's ID
                       const string name="ArrowedLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete an arrowed line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to create an arrowed line! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor points' values and set default values               |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowedLineEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an arrowed line
   if(!ArrowedLineCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line's anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the second anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2<accuracy-1)
         p2+=1;
      //--- move the point
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- half a second of delay
   Sleep(500);
//--- loop counter
   int h_steps=bars/2;
//--- move both anchor points horizontally at the same time
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- shift the points
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.03 seconds of delay
      Sleep(30);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete an arrowed line
   ArrowedLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_CHANNEL

Equidistant Channel



Note

For an equidistant channel, it is possible to specify the mode of its continuation to the right and/or to the left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly). The mode of filling the channel with color can also be set.

Example

The following script creates and moves an equidistant channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

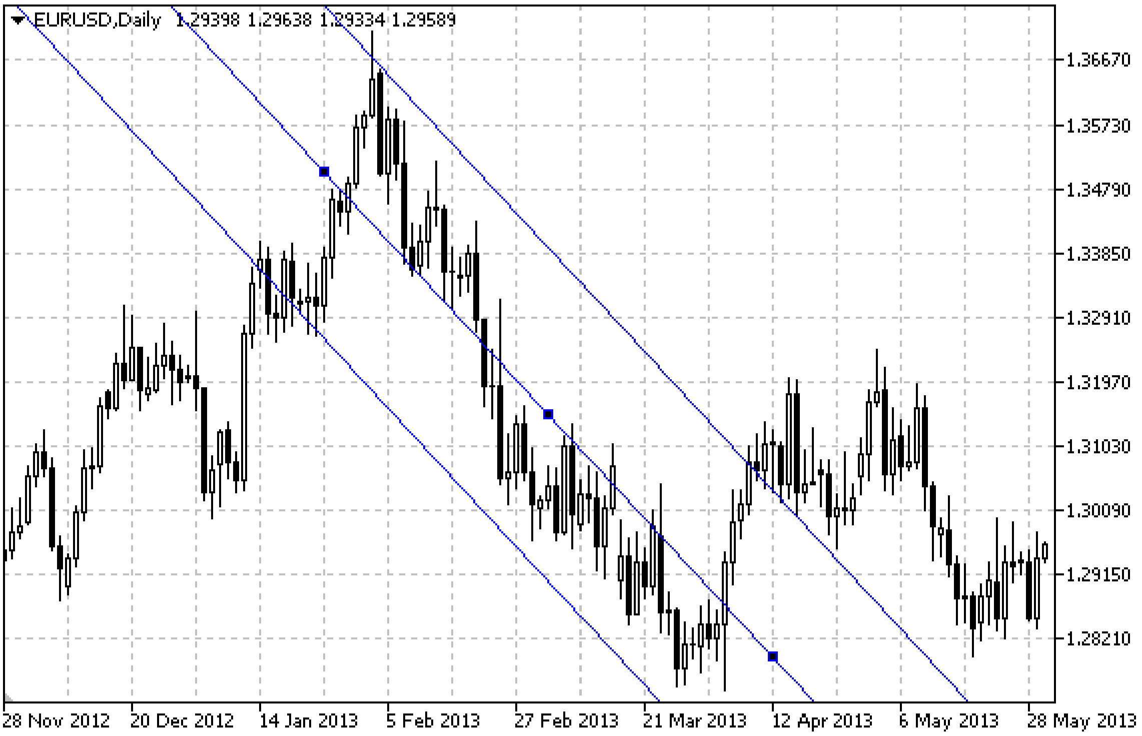
```
//--- description
#property description "Script draws \"Equidistant Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Channel";    // Channel name
input int             InpDate1=25;          // 1 st point's date, %
input int             InpPrice1=60;         // 1 st point's price, %
input int             InpDate2=65;          // 2 nd point's date, %
input int             InpPrice2=80;         // 2 nd point's price, %
input int             InpDate3=30;          // 3 rd point's date, %
input int             InpPrice3=40;         // 3 rd point's price, %
input color           InpColor=clrRed;      // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of channel lines
input int             InpWidth=2;           // Channel line width
input bool            InpBack=false;        // Background channel
input bool            InpFill=false;        // Filling the channel with color
input bool            InpSelection=true;    // Highlight to move
input bool            InpRayLeft=false;     // Channel's continuation to the left
input bool            InpRayRight=false;    // Channel's continuation to the right
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create an equidistant channel by the given coordinates           |
//+------------------------------------------------------------------+
bool ChannelCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="Channel",    // channel name
                   const int             sub_window=0,      // subwindow index
                   datetime              time1=0,           // first point time
                   double                price1=0,          // first point price
                   datetime              time2=0,           // second point time
                   double                price2=0,          // second point price
                   datetime              time3=0,           // third point time
                   double                price3=0,          // third point price
                   const color           clr=clrRed,        // channel color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel lines
                   const int             width=1,           // width of channel lines
                   const bool            fill=false,        // filling the channel wit
                   const bool            back=false,        // in the background
                   const bool            selection=true,    // highlight to move
                   const bool            ray_left=false,    // channel's continuation
                   const bool            ray_right=false,   // channel's continuation
                   const bool            hidden=true,       // hidden in the object li
                   const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time2,price2,tir
     {
      Print(__FUNCTION__,
            ": failed to create an equidistant channel! Error code = ",GetLastError()
      return(false);
     }
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the channel
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool ChannelPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="Channel", // channel name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time coordinate
                        double       price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool ChannelDelete(const long   chart_ID=0,     // chart's ID
                   const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+---------------------------------------------------------------------+
//| Check the values of the channel's anchor points and set default values  |
//| for empty ones                                                      |
//+---------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- if the second (right) point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points higher than the secon
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the first point's one
   if(!time3)
      time3=time1;
//--- if the third point's price is not set, it is equal to the second point's one
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create the equidistant channel
   if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
   int h_steps=bars/6;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1>1)
         d1-=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
```

```
   int v_steps=accuracy/10;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3>1)
         p3-=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   ChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_STDDEVCHANNEL

Standard Deviation Channel.



Note

For Standard Deviation Channel, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly). The mode of filling the channel with color can also be set.

OBJPROP_DEVIATION property is used to change the value of the channel deviation.

Example

The following script creates and moves Standard Deviation Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Standard Deviation Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="StdDevChannel";   // Channel name
input int             InpDate1=10;               // 1 st point's date, %
input int             InpDate2=40;               // 2 nd point's date, %
input double          InpDeviation=1.0;          // Deviation
input color           InpColor=clrRed;           // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int             InpWidth=2;                // Width of channel lines
input bool            InpFill=false;             // Filling the channel with color
input bool            InpBack=false;             // Background channel
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayLeft=false;          // Channel's continuation to the left
input bool            InpRayRight=false;         // Channel's continuation to the righ
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+-------------------------------------------------------------------+
//| Create standard deviation channel by the given coordinates       |
//+-------------------------------------------------------------------+
bool StdDevChannelCreate(const long            chart_ID=0,        // chart's ID
                         const string          name="Channel",    // channel name
                         const int             sub_window=0,      // subwindow index
                         datetime              time1=0,           // first point time
                         datetime              time2=0,           // second point time
                         const double          deviation=1.0,     // deviation
                         const color           clr=clrRed,        // channel color
                         const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel
                         const int             width=1,           // width of channel
                         const bool            fill=false,        // filling the chann
                         const bool            back=false,        // in the backgroun
                         const bool            selection=true,    // highlight to move
                         const bool            ray_left=false,    // channel's continu
                         const bool            ray_right=false,   // channel's continu
                         const bool            hidden=true,       // hidden in the obj
                         const long            z_order=0)         // priority for mous
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,time2);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create standard deviation channel! Error code = ",GetLastErro
      return(false);
     }
//--- set deviation value affecting the channel width
   ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the channel
```

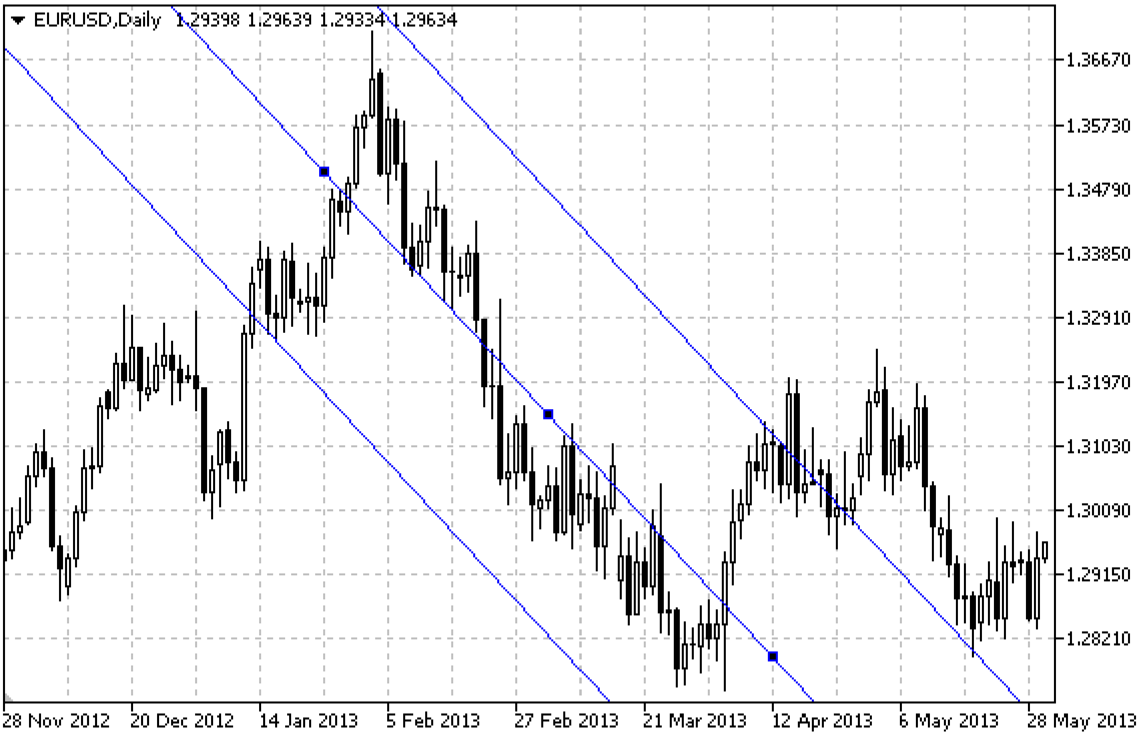
```
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool StdDevChannelPointChange(const long   chart_ID=0,      // chart's ID
                              const string name="Channel", // channel name
                              const int    point_index=0,  // anchor point index
                              datetime     time=0)         // anchor point time coordi
  {
//--- if point time is not set, move the point to the current bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the channel's deviation                                   |
//+------------------------------------------------------------------+
bool StdDevChannelDeviationChange(const long   chart_ID=0,      // chart's ID
                                  const string name="Channel", // channel name
                                  const double deviation=1.0)  // deviation
  {
//--- reset the error value
   ResetLastError();
//--- change trend line's slope angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
     {
      Print(__FUNCTION__,
            ": failed to change channel deviation! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool StdDevChannelDelete(const long   chart_ID=0,     // chart's ID
                         const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-------------------------------------------------------------------------+
//| Check the values of the channel's anchor points and set default values  |
//| for empty ones                                                          |
//+-------------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- create standard deviation channel
   if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpColor,InpStyl
      InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder
     {
      return;
     }
//--- redraw the chart and wait for 1 second
```

```
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel horizontally to the right and expand it
//--- loop counter
   int h_steps=bars/2;
//--- move the channel
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- move the anchor points
      if(!StdDevChannelPointChange(0,InpName,0,date[d1]))
         return;
      if(!StdDevChannelPointChange(0,InpName,1,date[d2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   double v_steps=InpDeviation*2;
//--- expand the channel
   for(double i=InpDeviation;i<v_steps;i+=10.0/accuracy)
     {
      if(!StdDevChannelDeviationChange(0,InpName,i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   StdDevChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
```

```
  }
```

OBJ_REGRESSION

Linear Regression Channel.


EURUSD,Daily 1.29398 1.29639 1.29334 1.29634
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
28 Nov 2012
20 Dec 2012
14 Jan 2013
5 Feb 2013
27 Feb 2013
21 Mar 2013
12 Apr 2013
6 May 2013
28 May 2013

Note

For Linear Regression Channel, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly). The mode of filling the channel with color can also be set.

Example

The following script creates and moves Linear Regression Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

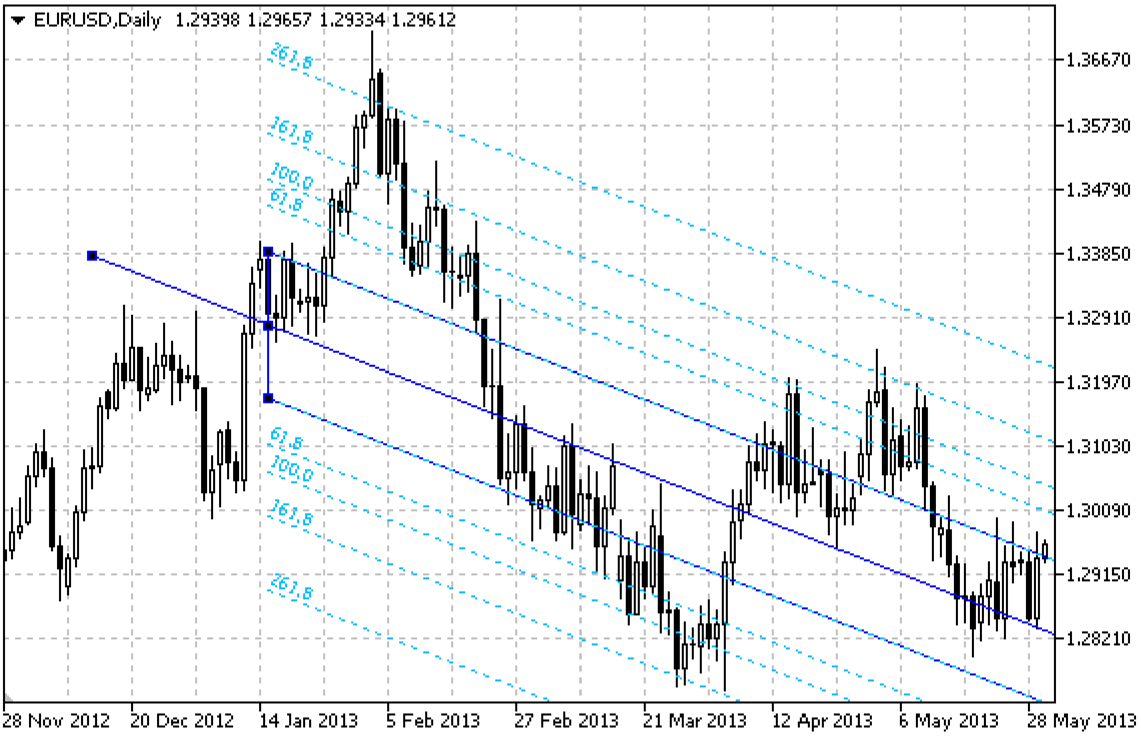
```
//--- description
#property description "Script draws \"Linear Regression Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Regression"; // Channel name
input int             InpDate1=10;          // 1 st point's date, %
input int             InpDate2=40;          // 2 nd point's date, %
input color           InpColor=clrRed;      // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;  // Style of channel lines
input int             InpWidth=2;           // Width of channel lines
input bool            InpFill=false;        // Filling the channel with color
input bool            InpBack=false;        // Background channel
input bool            InpSelection=true;    // Highlight to move
input bool            InpRayLeft=false;     // Channel's continuation to the left
input bool            InpRayRight=false;    // Channel's continuation to the right
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Linear Regression Channel by the given coordinates        |
//+------------------------------------------------------------------+
bool RegressionCreate(const long            chart_ID=0,        // chart's ID
                      const string          name="Regression", // channel name
                      const int             sub_window=0,      // subwindow index
                      datetime              time1=0,           // first point time
                      datetime              time2=0,           // second point time
                      const color           clr=clrRed,        // channel color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // style of channel li
                      const int             width=1,           // width of channel li
                      const bool            fill=false,        // filling the channel
                      const bool            back=false,        // in the background
                      const bool            selection=true,    // highlight to move
                      const bool            ray_left=false,    // channel's continuat
                      const bool            ray_right=false,   // channel's continuat
                      const bool            hidden=true,       // hidden in the object
                      const long            z_order=0)         // priority for mouse
  {
//--- set anchor points' coordinates if they are not set
   ChangeRegressionEmptyPoints(time1,time2);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create linear regression channel! Error code = ",GetLastErro
      return(false);
     }
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the channel
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
```

```
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool RegressionPointChange(const long   chart_ID=0,     // chart's ID
                           const string name="Channel", // channel name
                           const int    point_index=0,  // anchor point index
                           datetime     time=0)         // anchor point time coordinat
  {
//--- if point time is not set, move the point to the current bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool RegressionDelete(const long   chart_ID=0,     // chart's ID
                      const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of the channel's anchor points and set default values  |
//| for empty ones                                                         |
//+------------------------------------------------------------------+
void ChangeRegressionEmptyPoints(datetime &time1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
```

```
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- create linear regression channel
   if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,InpWidth,
      InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel horizontally to the right
//--- loop counter
   int h_steps=bars/2;
//--- move the channel
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- move the anchor points
      if(!RegressionPointChange(0,InpName,0,date[d1]))
         return;
      if(!RegressionPointChange(0,InpName,1,date[d2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   RegressionDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_PITCHFORK

Andrews’ Pitchfork.



Note

For Andrews’ Pitchfork, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Andrews’ Pitchfork on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Andrews’ Pitchfork\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Pitchfork";  // Pitchfork name
input int             InpDate1=14;          // 1 st point's date, %
input int             InpPrice1=40;         // 1 st point's price, %
input int             InpDate2=18;          // 2 nd point's date, %
input int             InpPrice2=50;         // 2 nd point's price, %
input int             InpDate3=18;          // 3 rd point's date, %
input int             InpPrice3=30;         // 3 rd point's price, %
input color           InpColor=clrRed;      // Pitchfork color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Style of pitchfork lines
input int             InpWidth=1;           // Width of pitchfork lines
input bool            InpBack=false;        // Background pitchfork
input bool            InpSelection=true;    // Highlight to move
input bool            InpRayLeft=false;     // Pitchfork's continuation to the left
input bool            InpRayRight=false;    // Pitchfork's continuation to the right
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Andrews' Pitchfork by the given coordinates               |
//+------------------------------------------------------------------+
bool PitchforkCreate(const long            chart_ID=0,        // chart's ID
                     const string          name="Pitchfork",  // pitchfork name
                     const int             sub_window=0,      // subwindow index
                     datetime              time1=0,           // first point time
                     double                price1=0,          // first point price
                     datetime              time2=0,           // second point time
                     double                price2=0,          // second point price
                     datetime              time3=0,           // third point time
                     double                price3=0,          // third point price
                     const color           clr=clrRed,        // color of pitchfork li
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // style of pitchfork li
                     const int             width=1,           // width of pitchfork li
                     const bool            back=false,        // in the background
                     const bool            selection=true,    // highlight to move
                     const bool            ray_left=false,    // pitchfork's continuat
                     const bool            ray_right=false,   // pitchfork's continuat
                     const bool            hidden=true,       // hidden in the object
                     const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create Andrews' Pitchfork by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create \"Andrews' Pitchfork\"! Error code = ",GetLastError())
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
```

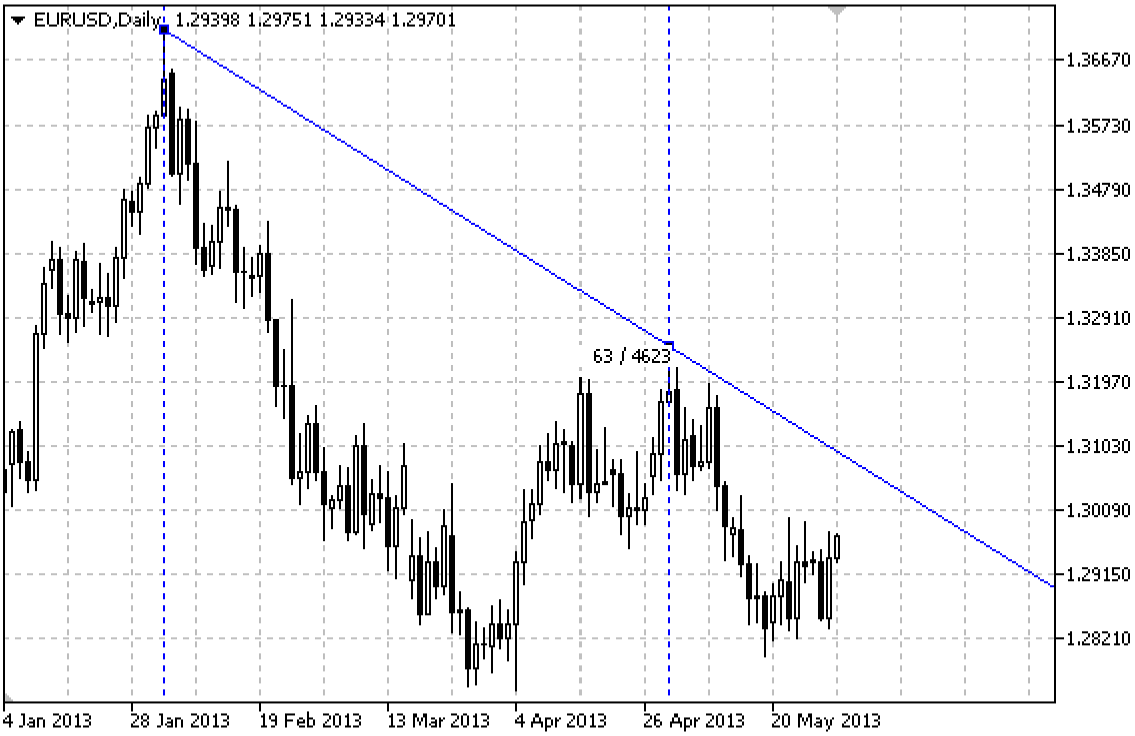
```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the pitchfork for movi
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the pitchfork's dis
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the pitchfork's dis
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of Andrews' Pitchfork levels and their parameters     |
//+------------------------------------------------------------------+
bool PitchforkLevelsSet(int             levels,           // number of level lines
                        double          &values[],        // values of level lines
                        color           &colors[],        // color of level lines
                        ENUM_LINE_STYLE &styles[],        // style of level lines
                        int             &widths[],        // width of level lines
                        const long      chart_ID=0,       // chart's ID
                        const string    name="Pitchfork") // pitchfork name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Andrews' Pitchfork anchor point                             |
//+------------------------------------------------------------------+
bool PitchforkPointChange(const long   chart_ID=0,       // chart's ID
                          const string name="Pitchfork", // channel name
                          const int    point_index=0,    // anchor point index
                          datetime     time=0,           // anchor point time coordina
                          double       price=0)          // anchor point price coordin
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Andrews’ Pitchfork                                        |
//+------------------------------------------------------------------+
bool PitchforkDelete(const long   chart_ID=0,       // chart's ID
                     const string name="Pitchfork") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Andrews' Pitchfork\"! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Andrews' Pitchfork anchor points and set default |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- if the second (upper right) point's time is not set, it will be on the current
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the second one
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second point's one
   if(!time3)
      time3=time2;
//--- if the third point's price is not set, move it 200 points lower than the first
   if(!price3)
      price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
```

```
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Andrews' Pitchfork anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Andrews' Pitchfork
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create the pitchfork
   if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pric
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the pitchfork's anchor points
//--- loop counter
   int v_steps=accuracy/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
```

```
         if(!PitchforkPointChange(0,InpName,0,date[d1],price[p1]))
            return;
         //--- check if the script's operation has been forcefully disabled
         if(IsStopped())
            return;
         //--- redraw the chart
         ChartRedraw();
        }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/8;
//--- move the third anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d3<bars-1)
         d3+=1;
      //--- move the point
      if(!PitchforkPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/10;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!PitchforkPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the pitchfork from the chart
   PitchforkDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNLINE

Gann Line.



Note

For Gann Line, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

Both Gann angle with a scale and coordinates of the second anchor point can be used to set the slope of the line.

Example

The following script creates and moves Gann Line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

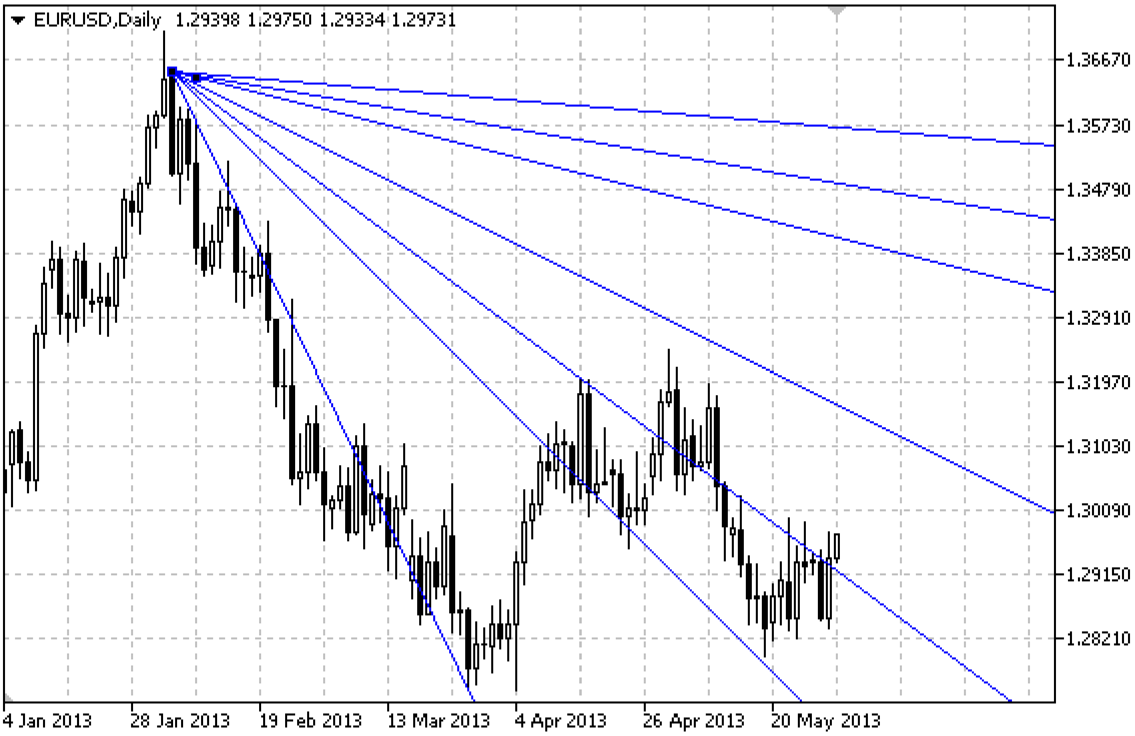
```
//--- description
#property description "Script draws \"Gann Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannLine";        // Line name
input int             InpDate1=20;               // 1 st point's date, %
input int             InpPrice1=75;              // 1 st point's price, %
input int             InpDate2=80;               // 2 nd point's date, %
input double          InpAngle=0.0;              // Gann Angle
input double          InpScale=1.0;              // Scale
input color           InpColor=clrRed;           // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=2;                // Line width
input bool            InpBack=false;             // Background line
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayLeft=false;          // Line's continuation to the left
input bool            InpRayRight=true;          // Line's continuation to the right
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Gann Line by the coordinates, angle and scale             |
//+------------------------------------------------------------------+
bool GannLineCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="GannLine",   // line name
                    const int             sub_window=0,      // subwindow index
                    datetime              time1=0,           // first point time
                    double                price1=0,          // first point price
                    datetime              time2=0,           // second point time
                    const double          angle=1.0,         // Gann angle
                    const double          scale=1.0,         // scale
                    const color           clr=clrRed,        // line color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                    const int             width=1,           // line width
                    const bool            back=false,        // in the background
                    const bool            selection=true,    // highlight to move
                    const bool            ray_left=false,    // line's continuation t
                    const bool            ray_right=true,    // line's continuation t
                    const bool            hidden=true,       // hidden in the object l
                    const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannLineEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Line by the given coordinates
//--- correct coordinate of the second anchor point is redefined
//--- automatically after Gann angle and/or the scale changes,
   if(!ObjectCreate(chart_ID,name,OBJ_GANNLINE,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Line\"! Error code = ",GetLastError());
      return(false);
     }
//--- change Gann angle
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- set line color
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the lines for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the line's display
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Line anchor point                                      |
//+------------------------------------------------------------------+
bool GannLinePointChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannLine", // line name
                         const int    point_index=0,   // anchor point index
                         datetime     time=0,          // anchor point time coordinate
                         double       price=0)         // anchor point price coordinat
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the line's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann angle                                                |
//+------------------------------------------------------------------+
bool GannLineAngleChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannLine", // line name
                         const double angle=1.0)       // Gann angle
  {
//--- reset the error value
   ResetLastError();
//--- change Gann angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change Gann angle! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Line's scale                                         |
//+------------------------------------------------------------------+
bool GannLineScaleChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannLine", // line name
                         const double scale=1.0)       // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Line from the chart                    |
//+------------------------------------------------------------------+
bool GannLineDelete(const long   chart_ID=0,      // chart's ID
                    const string name="GannLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann line
   if(!ObjectDelete(chart_ID,name))
```

```
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Line\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Gann Line anchor points and set default      |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Line
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Line
   if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,InpScale,InpCol
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line's anchor point and change the angle
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- half a second of delay
   Sleep(500);
//--- define the current value of Gann angle (changed
//--- after moving the first anchor point)
   double curr_angle;
   if(!ObjectGetDouble(0,InpName,OBJPROP_ANGLE,0,curr_angle))
      return;
//--- loop counter
   v_steps=accuracy/8;
//--- change Gann angle
   for(int i=0;i<v_steps;i++)
     {
      if(!GannLineAngleChange(0,InpName,curr_angle-0.05*i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the line from the chart
   GannLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNFAN

Gann Fan.



Note

For Gann Fan, it is possible to specify trend type from ENUM_GANN_DIRECTION enumeration. By adjusting the scale value (OBJPROP_SCALE), it is possible to change slope angle of the fan lines.

Example

The following script creates and moves Gann Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Gann Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannFan";         // Fan name
input int             InpDate1=15;               // 1 st point's date, %
input int             InpPrice1=25;              // 1 st point's price, %
input int             InpDate2=85;               // 2 nd point's date, %
input double          InpScale=2.0;              // Scale
input bool            InpDirection=false;        // Trend direction
input color           InpColor=clrRed;           // Fan color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of fan lines
input int             InpWidth=1;                // Width of fan lines
input bool            InpBack=false;             // Background fan
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Gann Fan                                                  |
//+------------------------------------------------------------------+
bool GannFanCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="GannFan",    // fan name
                   const int             sub_window=0,      // subwindow index
                   datetime              time1=0,           // first point time
                   double                price1=0,          // first point price
                   datetime              time2=0,           // second point time
                   const double          scale=1.0,         // scale
                   const bool            direction=true,    // trend direction
                   const color           clr=clrRed,        // fan color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of fan lines
                   const int             width=1,           // width of fan lines
                   const bool            back=false,        // in the background
                   const bool            selection=true,    // highlight to move
                   const bool            hidden=true,       // hidden in the object li
                   const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannFanEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Fan by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- change Gann Fan's trend direction (true - descending, false - ascending)
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- set fan color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the fan lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the fan lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the fan for moving
```

```
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Fan anchor point                                       |
//+------------------------------------------------------------------+
bool GannFanPointChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannFan", // fan name
                        const int    point_index=0,   // anchor point index
                        datetime     time=0,          // anchor point time coordinate
                        double       price=0)         // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the fan's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Fan's scale                                          |
//+------------------------------------------------------------------+
bool GannFanScaleChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannFan", // fan name
                        const double scale=1.0)       // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
```

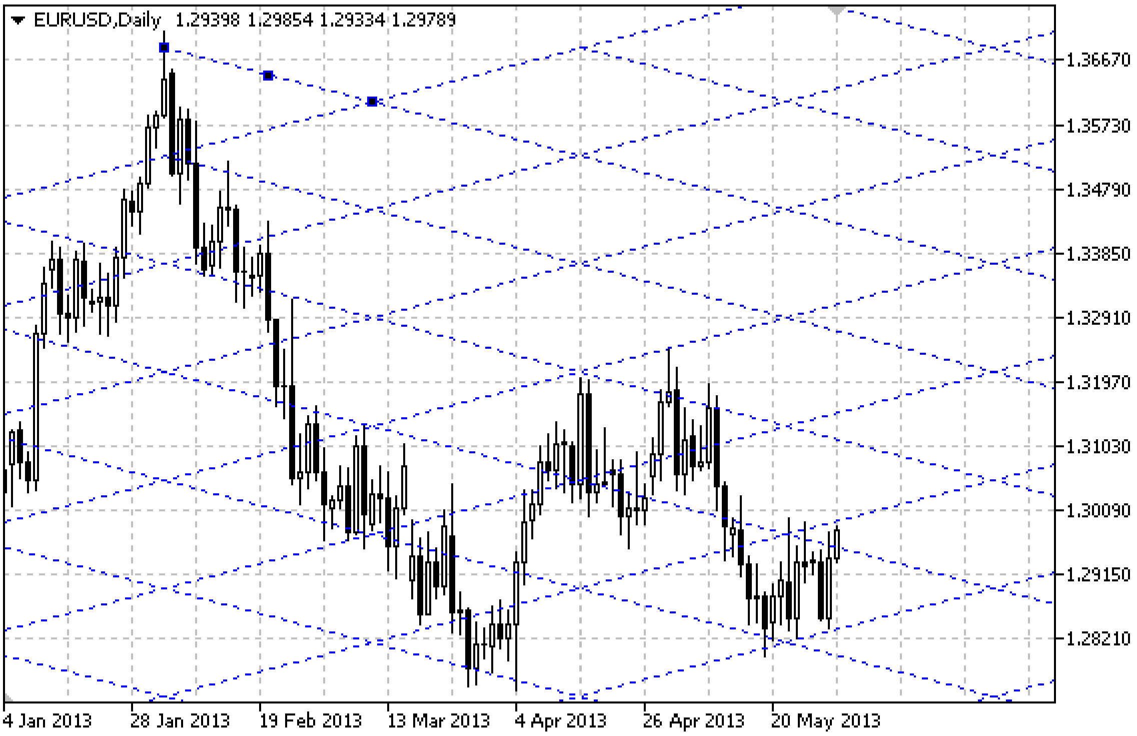
```
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Fan's trend direction                                |
//+------------------------------------------------------------------+
bool GannFanDirectionChange(const long   chart_ID=0,     // chart's ID
                            const string name="GannFan", // fan name
                            const bool   direction=true) // trend direction
  {
//--- reset the error value
   ResetLastError();
//--- change Gann Fan's trend direction
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Fan from the chart                     |
//+------------------------------------------------------------------+
bool GannFanDelete(const long   chart_ID=0,     // chart's ID
                   const string name="GannFan") // fan name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann Fan
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
// | Check the values of Gann Fan anchor points and set default      |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of fan's anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
```

```
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Fan
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Fan
   if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the fan's anchor point
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      //--- move the point
      if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change fan's trend direction to descending one
   GannFanDirectionChange(0,InpName,true);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the fan from the chart
   GannFanDelete(0,InpName);
   ChartRedraw();
```

```
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNGRID

Gann Grid.



Note

For Gann Grid, it is possible to specify trend type from ENUM_GANN_DIRECTION. By adjusting the scale value (OBJPROP_SCALE), it is possible to change slope angle of the grid lines.

Example

The following script creates and moves Gann Grid on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Gann Grid\" graphical object."
#property description "Anchor point coordinates of the grid are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannGrid";        // Grid name
input int             InpDate1=15;               // 1 st point's date, %
input int             InpPrice1=25;              // 1 st point's price, %
input int             InpDate2=35;               // 2 nd point's date, %
input double          InpScale=3.0;              // Scale
input bool            InpDirection=false;        // Trend direction
input color           InpColor=clrRed;           // Grid color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of grid lines
input int             InpWidth=1;                // Width of fan lines
input bool            InpBack=false;             // Background grid
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Gann Grid                                                 |
//+------------------------------------------------------------------+
bool GannGridCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="GannGrid",   // grid name
                    const int             sub_window=0,      // subwindow index
                    datetime              time1=0,           // first point time
                    double                price1=0,          // first point price
                    datetime              time2=0,           // second point time
                    const double          scale=1.0,         // scale
                    const bool            direction=true,    // trend direction
                    const color           clr=clrRed,        // grid color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of grid lines
                    const int             width=1,           // width of grid lines
                    const bool            back=false,        // in the background
                    const bool            selection=true,    // highlight to move
                    const bool            hidden=true,       // hidden in the object
                    const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannGridEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Grid by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Grid\"! Error code = ",GetLastError());
      return(false);
     }
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- change Gann Fan's trend direction (true - descending, false - ascending)
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- set grid color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the grid lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the grid lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the grid for moving
```

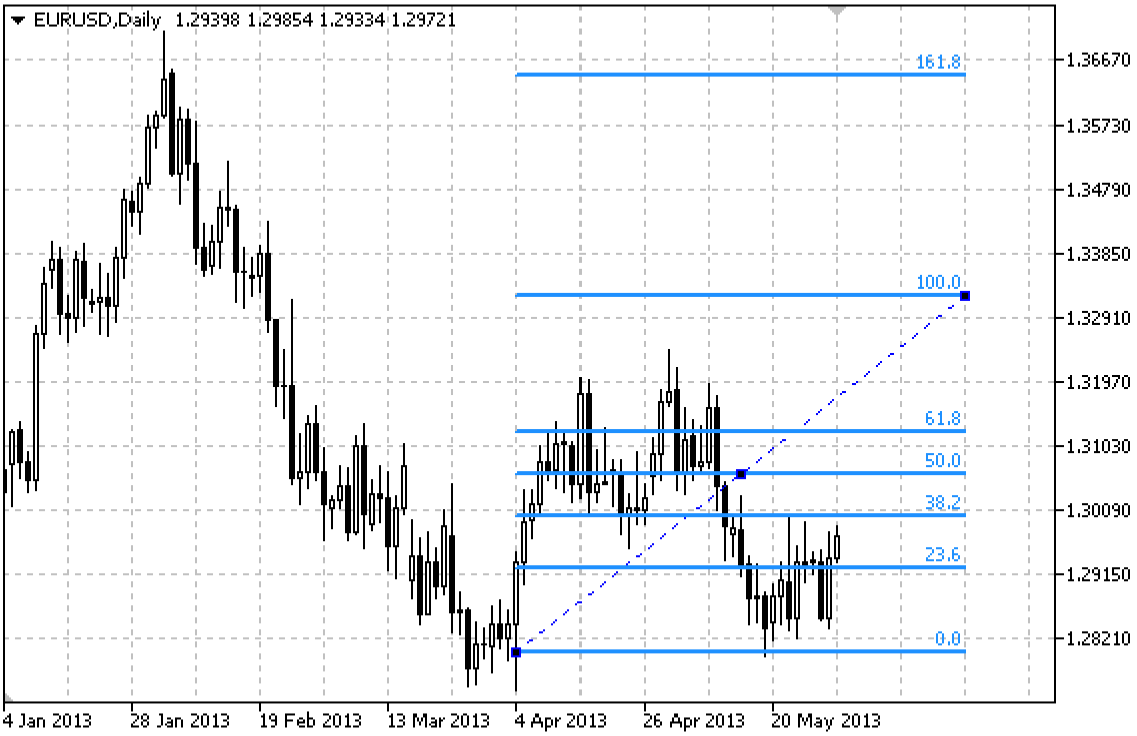
```
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Grid anchor point                                      |
//+------------------------------------------------------------------+
bool GannGridPointChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannGrid", // grid name
                         const int    point_index=0,   // anchor point index
                         datetime     time=0,          // anchor point time coordinate
                         double       price=0)         // anchor point price coordinat
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the grid's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Grid's scale                                         |
//+------------------------------------------------------------------+
bool GannGridScaleChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannGrid", // grids
                         const double scale=1.0)       // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
```

```
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Grid's trend direction                               |
//+------------------------------------------------------------------+
bool GannGridDirectionChange(const long   chart_ID=0,      // chart's ID
                             const string name="GannGrid", // grid name
                             const bool   direction=true)  // trend direction
  {
//--- reset the error value
   ResetLastError();
//--- change Gann Grid's trend direction
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Fan from the chart                     |
//+------------------------------------------------------------------+
bool GannGridDelete(const long   chart_ID=0,      // chart's ID
                    const string name="GannGrid") // grid name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann Grid
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Grid\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Gann Grid anchor points and set default      |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing grid anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
```

```
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Grid
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Grid
   if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the grid's anchor points
//--- loop counter
   int v_steps=accuracy/4;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/4;
//--- move the second anchor point horizontally
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      if(!GannGridPointChange(0,InpName,1,date[d2],0))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change grid's trend direction to descending one
   GannGridDirectionChange(0,InpName,true);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the grid from the chart
   GannGridDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBO

Fibonacci Retracement.



Note

For Fibonacci Retracement, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Retracement on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Retracement\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboLevels";      // Object name
input int             InpDate1=10;               // 1 st point's date, %
input int             InpPrice1=65;              // 1 st point's price, %
input int             InpDate2=90;               // 2 nd point's date, %
input int             InpPrice2=85;              // 2 nd point's price, %
input color           InpColor=clrRed;           // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=2;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayLeft=false;          // Object's continuation to the left
input bool            InpRayRight=false;         // Object's continuation to the righ
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Retracement by the given coordinates            |
//+------------------------------------------------------------------+
bool FiboLevelsCreate(const long            chart_ID=0,        // chart's ID
                      const string          name="FiboLevels", // object name
                      const int             sub_window=0,      // subwindow index
                      datetime              time1=0,           // first point time
                      double                price1=0,          // first point price
                      datetime              time2=0,           // second point time
                      double                price2=0,          // second point price
                      const color           clr=clrRed,        // object color
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                      const int             width=1,           // object line width
                      const bool            back=false,        // in the background
                      const bool            selection=true,    // highlight to move
                      const bool            ray_left=false,    // object's continuati
                      const bool            ray_right=false,   // object's continuati
                      const bool            hidden=true,       // hidden in the object
                      const long            z_order=0)         // priority for mouse
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- Create Fibonacci Retracement by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Retracement\"! Error code = ",GetLastErro
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
```

```
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the object's displa
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the object's displa
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboLevelsSet(int             levels,            // number of level lines
                   double          &values[],         // values of level lines
                   color           &colors[],         // color of level lines
                   ENUM_LINE_STYLE &styles[],         // style of level lines
                   int             &widths[],         // width of level lines
                   const long      chart_ID=0,        // chart's ID
                   const string    name="FiboLevels") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) || levels!=ArraySize(widt
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- successful execution
```

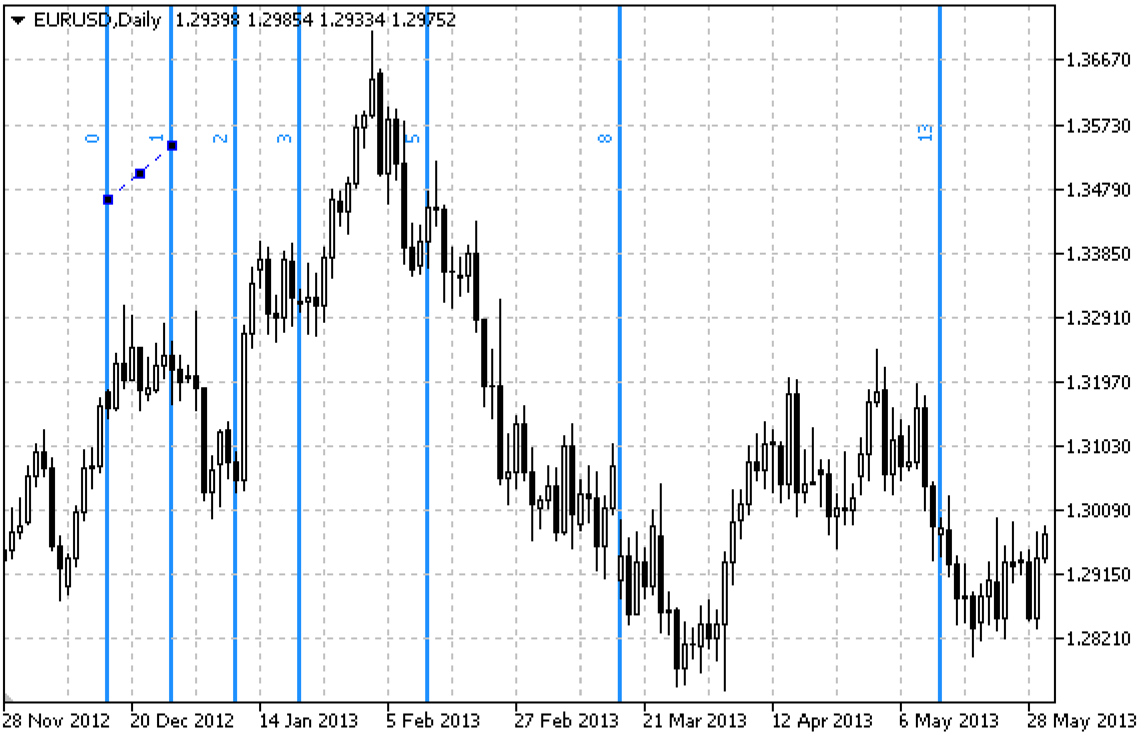
```
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Retracement anchor point                          |
//+------------------------------------------------------------------+
bool FiboLevelsPointChange(const long   chart_ID=0,        // chart's ID
                           const string name="FiboLevels", // object name
                           const int    point_index=0,     // anchor point index
                           datetime     time=0,            // anchor point time coordi
                           double       price=0)           // anchor point price coord
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Retracement                                     |
//+------------------------------------------------------------------+
bool FiboLevelsDelete(const long   chart_ID=0,        // chart's ID
                      const string name="FiboLevels") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Retracement\"! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Retracement anchor points and set  |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                 datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the second one
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Retracement anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
```

```
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Retracement
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy*2/5;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the second anchor point
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboLevelsDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOTIMES

Fibonacci Time Zones.



Note

For "Fibonacci Time Zones", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Time Zones on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Time Zones\" graphical object."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboTimes";       // Object name
input int             InpDate1=10;               // 1 st point's date, %
input int             InpPrice1=45;              // 1 st point's price, %
input int             InpDate2=20;               // 2 nd point's date, %
input int             InpPrice2=55;              // 2 nd point's price, %
input color           InpColor=clrRed;           // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=2;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Time Zones by the given coordinates             |
//+------------------------------------------------------------------+
bool FiboTimesCreate(const long            chart_ID=0,        // chart's ID
                     const string          name="FiboTimes",  // object name
                     const int             sub_window=0,      // subwindow index
                     datetime              time1=0,           // first point time
                     double                price1=0,          // first point price
                     datetime              time2=0,           // second point time
                     double                price2=0,          // second point price
                     const color           clr=clrRed,        // object color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // object line style
                     const int             width=1,           // object line width
                     const bool            back=false,        // in the background
                     const bool            selection=true,    // highlight to move
                     const bool            hidden=true,       // hidden in the object
                     const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboTimesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Time Zones by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Time Zones\"! Error code = ",GetLastError
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
```

```
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboTimesLevelsSet(int             levels,           // number of level lines
                        double          &values[],        // values of level lines
                        color           &colors[],        // color of level lines
                        ENUM_LINE_STYLE &styles[],        // style of level lines
                        int             &widths[],        // width of level lines
                        const long      chart_ID=0,       // chart's ID
                        const string    name="FiboTimes") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(values[i],1));
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Time Zones anchor point                           |
//+------------------------------------------------------------------+
bool FiboTimesPointChange(const long   chart_ID=0,       // chart's ID
                          const string name="FiboTimes", // object name
                          const int    point_index=0,    // anchor point index
                          datetime     time=0,           // anchor point time coordina
                          double       price=0)          // anchor point price coordi
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Time Zones                                      |
//+------------------------------------------------------------------+
bool FiboTimesDelete(const long   chart_ID=0,       // chart's ID
                     const string name="FiboTimes") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Time Zones\"! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Time Zones and                     |
//| set default values for empty ones                                |
//+------------------------------------------------------------------+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
```

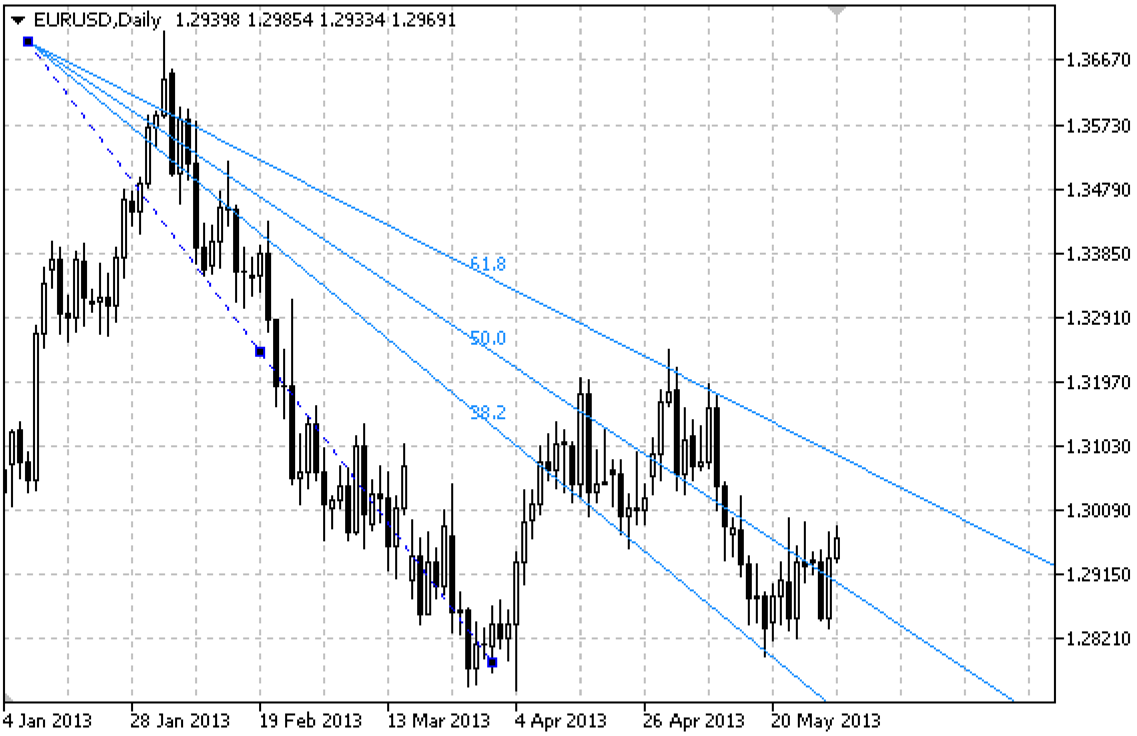
```
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 2 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 3 bars
      datetime temp[3];
      CopyTime(Symbol(),Period(),time1,3,temp);
      //--- set the first point 2 bars left from the second one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first point's one
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Time Zones anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
```

```
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Time Zones
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboTimesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int h_steps=bars*2/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboTimesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   h_steps=bars*3/5;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1<bars-1)
         d1+=1;
      //--- move the point
      if(!FiboTimesPointChange(0,InpName,0,date[d1],price[p1]))
```

```
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboTimesDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOFAN

Fibonacci Fan.



Note

For "Fibonacci Fan", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboFan";          // Fan name
input int             InpDate1=10;                // 1 st point's date, %
input int             InpPrice1=25;               // 1 st point's price, %
input int             InpDate2=30;                // 2 nd point's date, %
input int             InpPrice2=50;               // 2 nd point's price, %
input color           InpColor=clrRed;            // Fan line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT;  // Line style
input int             InpWidth=2;                 // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Fan by the given coordinates                    |
//+------------------------------------------------------------------+
bool FiboFanCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="FiboFan",    // fan name
                   const int             sub_window=0,      // subwindow index
                   datetime              time1=0,           // first point time
                   double                price1=0,          // first point price
                   datetime              time2=0,           // second point time
                   double                price2=0,          // second point price
                   const color           clr=clrRed,        // fan line color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // fan line style
                   const int             width=1,           // fan line width
                   const bool            back=false,        // in the background
                   const bool            selection=true,    // highlight to move
                   const bool            hidden=true,       // hidden in the object l
                   const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Fan by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the fan for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
```

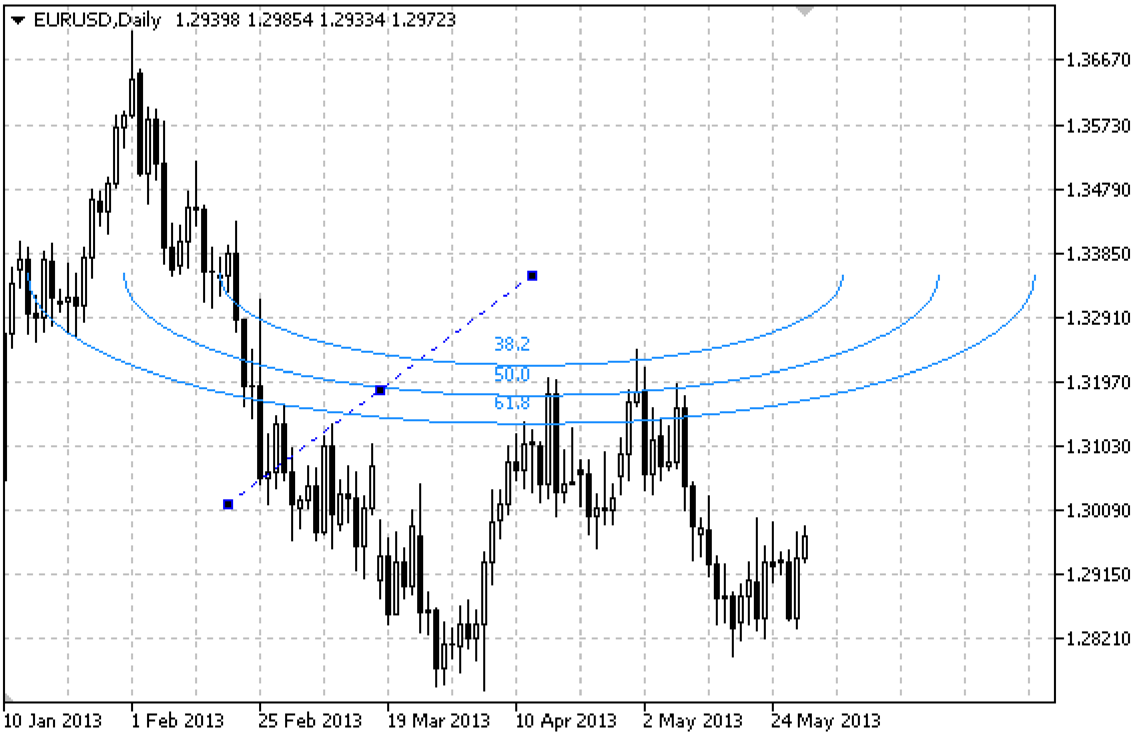
```
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboFanLevelsSet(int             levels,         // number of level lines
                      double          &values[],      // values of level lines
                      color           &colors[],      // color of level lines
                      ENUM_LINE_STYLE &styles[],      // style of level lines
                      int             &widths[],      // width of level lines
                      const long      chart_ID=0,     // chart's ID
                      const string    name="FiboFan") // fan name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Fan anchor point                                  |
//+------------------------------------------------------------------+
bool FiboFanPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="FiboFan", // fan name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time coordinate
                        double        price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Fan                                             |
//+------------------------------------------------------------------+
bool FiboFanDelete(const long   chart_ID=0,     // chart's ID
                   const string name="FiboFan") // fan name
  {
//--- reset the error value
   ResetLastError();
//--- delete the fan
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Fan\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Fan anchor points and set          |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
```

```
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the second one
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Fan anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
```

```
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Fan
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboFanCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the fan's anchor points
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      //--- move the point
      if(!FiboFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/4;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboFanPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboFanDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOARC

Fibonacci Arcs.



Note

For "Fibonacci Arcs", it is possible to specify the display mode of the entire ellipse. Curvature radius can be specified by changing the scale and coordinates of the anchor points.

You can also specify the number of line-levels, their values and color.

Example

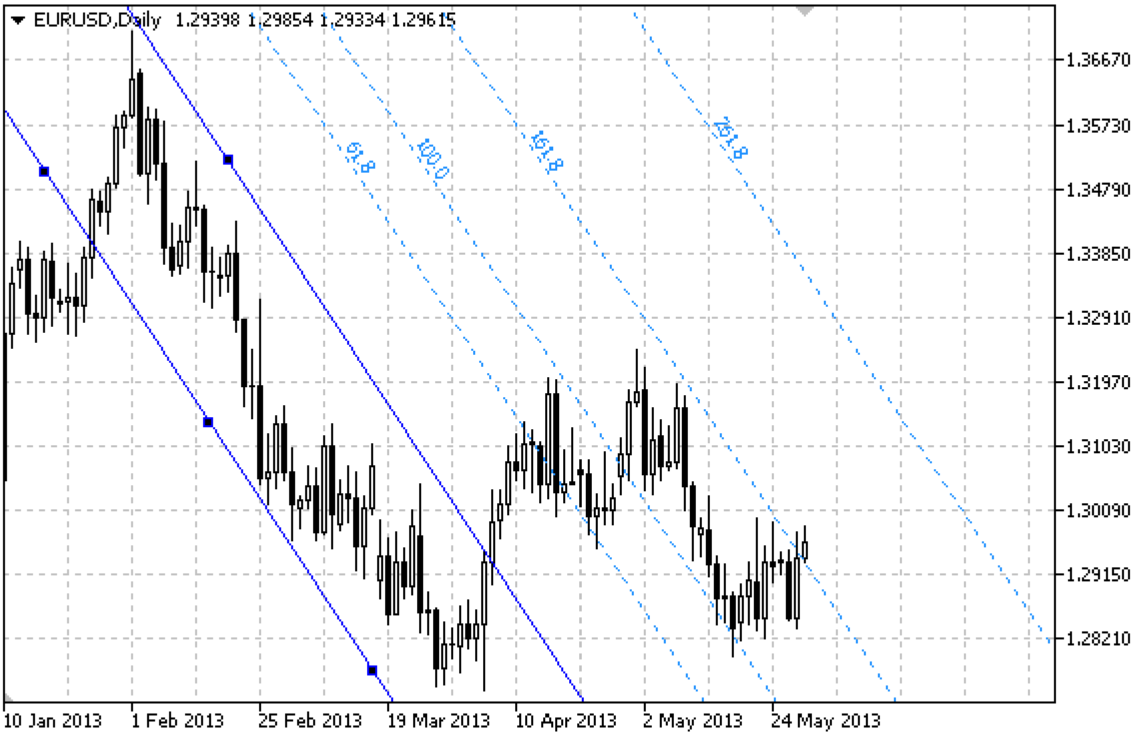
The following script creates and moves Fibonacci Arcs on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Arcs\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboArc";          // Object name
input int             InpDate1=25;                // 1 st point's date, %
input int             InpPrice1=25;               // 1 st point's price, %
input int             InpDate2=35;                // 2 nd point's date, %
input int             InpPrice2=55;               // 2 nd point's price, %
input double          InpScale=3.0;               // Scale
input bool            InpFullEllipse=true;       // Shape of the arcs
input color           InpColor=clrRed;           // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=2;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Arcs by the given coordinates                   |
//+------------------------------------------------------------------+
bool FiboArcCreate(const long            chart_ID=0,         // chart's ID
                   const string          name="FiboArc",     // object name
                   const int             sub_window=0,       // subwindow index
                   datetime              time1=0,            // first point time
                   double                price1=0,           // first point price
                   datetime              time2=0,            // second point time
                   double                price2=0,           // second point price
                   const double          scale=1.0,          // scale
                   const bool            full_ellipse=false, // shape of the arcs
                   const color           clr=clrRed,         // line color
                   const ENUM_LINE_STYLE style=STYLE_SOLID,  // line style
                   const int             width=1,            // line width
                   const bool            back=false,         // in the background
                   const bool            selection=true,     // highlight to move
                   const bool            hidden=true,        // hidden in the object
                   const long            z_order=0)          // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Arcs by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Arcs\"! Error code = ",GetLastError());
      return(false);
     }
//--- set the scale
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- set display of the arcs as a full ellipse (true) or a half of it (false)
   ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the arcs for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboArcLevelsSet(int             levels,         // number of level lines
                      double          &values[],      // values of level lines
                      color           &colors[],      // color of level lines
                      ENUM_LINE_STYLE &styles[],      // style of level lines
                      int             &widths[],      // width of level lines
                      const long      chart_ID=0,     // chart's ID
                      const string    name="FiboArc") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Arcs anchor point                                 |
//+------------------------------------------------------------------+
bool FiboArcPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="FiboArc", // object name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time coordinate
                        double       price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Arcs                                            |
//+------------------------------------------------------------------+
bool FiboArcDelete(const long   chart_ID=0,     // chart's ID
                   const string name="FiboArc") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Arcs\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Arcs anchor points and set default |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from the second
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points below the second one
   if(!price1)
      price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Arcs anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Arcs
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpScale,
      InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrc
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      //--- move the point
      if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboArcPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboArcDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOCHANNEL

Fibonacci Channel.



Note

For Fibonacci Channel, it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Channel\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboChannel";     // Channel name
input int             InpDate1=20;               // 1 st point's date, %
input int             InpPrice1=10;              // 1 st point's price, %
input int             InpDate2=60;               // 2 nd point's date, %
input int             InpPrice2=30;              // 2 nd point's price, %
input int             InpDate3=20;               // 3 rd point's date, %
input int             InpPrice3=25;              // 3 rd point's price, %
input color           InpColor=clrRed;           // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int             InpWidth=2;                // Width of channel lines
input bool            InpBack=false;             // Background channel
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayLeft=false;          // Channel's continuation to the lef
input bool            InpRayRight=false;         // Channel's continuation to the righ
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Channel by the given coordinates                |
//+------------------------------------------------------------------+
bool FiboChannelCreate(const long            chart_ID=0,         // chart's ID
                       const string          name="FiboChannel", // channel name
                       const int             sub_window=0,       // subwindow index
                       datetime              time1=0,            // first point time
                       double                price1=0,           // first point price
                       datetime              time2=0,            // second point time
                       double                price2=0,           // second point pric
                       datetime              time3=0,            // third point time
                       double                price3=0,           // third point price
                       const color           clr=clrRed,         // channel color
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // style of channel l
                       const int             width=1,            // width of channel l
                       const bool            back=false,         // in the background
                       const bool            selection=true,     // highlight to move
                       const bool            ray_left=false,     // channel's continua
                       const bool            ray_right=false,    // channel's continua
                       const bool            hidden=true,        // hidden in the obje
                       const long            z_order=0)          // priority for mouse
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Channel\"! Error code = ",GetLastError());
      return(false);
     }
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the channel's displ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboChannelLevelsSet(int             levels,             // number of level line
                          double          &values[],          // values of level line
                          color           &colors[],          // color of level lines
                          ENUM_LINE_STYLE &styles[],          // style of level lines
                          int             &widths[],          // width of level lines
                          const long      chart_ID=0,         // chart's ID
                          const string    name="FiboChannel") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Channel anchor point                              |
//+------------------------------------------------------------------+
bool FiboChannelPointChange(const long   chart_ID=0,         // chart's ID
                            const string name="FiboChannel", // channel name
                            const int    point_index=0,      // anchor point index
                            datetime     time=0,             // anchor point time coor
                            double       price=0)            // anchor point price coo
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool FiboChannelDelete(const long   chart_ID=0,         // chart's ID
                       const string name="FiboChannel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Channel\"! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
```

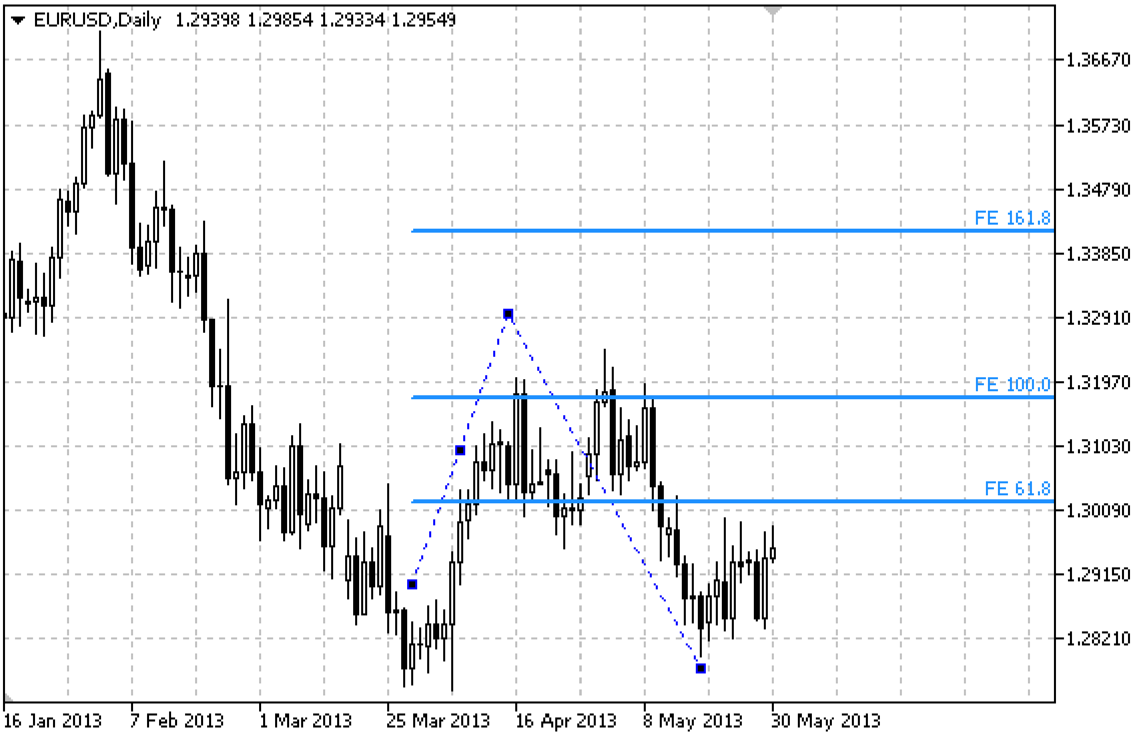
```
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Channel anchor points and set      |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                  double &price2,datetime &time3,double &price3)
  {
//--- if the second (right) point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points higher than the second
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the first point's one
   if(!time3)
      time3=time1;
//--- if the third point's price is not set, it is equal to the second point's one
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

```
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Channel
   if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
   int h_steps=bars/10;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1>1)
         d1-=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/10;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/15;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3<accuracy-1)
         p3+=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
```

```
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   FiboChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_EXPANSION

Fibonacci Expansion.



Note

For "Fibonacci Expansion", it is possible to specify the mode of continuation of its display to the right and/or left (OBJPROP_RAY_RIGHT and OBJPROP_RAY_LEFT properties accordingly).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Expansion on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Fibonacci Expansion\"graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboExpansion";   // Object name
input int             InpDate1=10;               // 1 st point's date, %
input int             InpPrice1=55;              // 1 st point's price, %
input int             InpDate2=30;               // 2 nd point's date, %
input int             InpPrice2=10;              // 2 nd point's price, %
input int             InpDate3=80;               // 3 rd point's date, %
input int             InpPrice3=75;              // 3 rd point's price, %
input color           InpColor=clrRed;           // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of lines
input int             InpWidth=2;                // Width of the lines
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayLeft=false;          // Object's continuation to the left
input bool            InpRayRight=false;         // Object's continuation to the right
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Fibonacci Extension by the given coordinates               |
//+------------------------------------------------------------------+
bool FiboExpansionCreate(const long            chart_ID=0,           // chart's ID
                         const string          name="FiboExpansion", // channel name
                         const int             sub_window=0,         // subwindow inde
                         datetime              time1=0,              // first point ti
                         double                price1=0,             // first point pr
                         datetime              time2=0,              // second point t
                         double                price2=0,             // second point p
                         datetime              time3=0,              // third point ti
                         double                price3=0,             // third point pr
                         const color           clr=clrRed,           // object color
                         const ENUM_LINE_STYLE style=STYLE_SOLID,    // style of the l
                         const int             width=1,              // width of the l
                         const bool            back=false,           // in the backgro
                         const bool            selection=true,       // highlight to m
                         const bool            ray_left=false,       // object's conti
                         const bool            ray_right=false,      // object's conti
                         const bool            hidden=true,          // hidden in the
                         const long            z_order=0)            // priority for m
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- Create Fibonacci Extension by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Extension\"! Error code = ",GetLastError()
      return(false);
     }
//--- set the object's color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the object's visual
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- enable (true) or disable (false) the mode of continuation of the object's visual
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboExpansionLevelsSet(int             levels,               // number of level l
                            double          &values[],            // values of level l
                            color           &colors[],            // color of level li
                            ENUM_LINE_STYLE &styles[],            // style of level li
                            int             &widths[],            // width of level li
                            const long      chart_ID=0,           // chart's ID
                            const string    name="FiboExpansion") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number of levels,
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToString(100*value
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Expansion anchor point                            |
//+------------------------------------------------------------------+
bool FiboExpansionPointChange(const long   chart_ID=0,           // chart's ID
                              const string name="FiboExpansion", // object name
                              const int    point_index=0,        // anchor point index
                              datetime     time=0,               // anchor point time
                              double       price=0)              // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Expansion                                       |
//+------------------------------------------------------------------+
bool FiboExpansionDelete(const long   chart_ID=0,           // chart's ID
                         const string name="FiboExpansion") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Expansion\"! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Expansion anchor points and set    |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                    double &price2,datetime &time3,double &price3)
  {
//--- if the third (right) point's time is not set, it will be on the current bar
   if(!time3)
      time3=TimeCurrent();
//--- if the third point's price is not set, it will have Bid value
   if(!price3)
      price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars left from the
//--- array for receiving the open time of the last 10 bars
   datetime temp[];
   ArrayResize(temp,10);
   if(!time1)
     {
      CopyTime(Symbol(),Period(),time3,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it is equal to the third point's one
   if(!price1)
      price1=price3;
//--- if the second point's time is not set, it is located 7 bars left from the third
   if(!time2)
      time2=temp[2];
//--- if the second point's price is not set, move it 250 points lower than the first
   if(!price2)
      price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
```

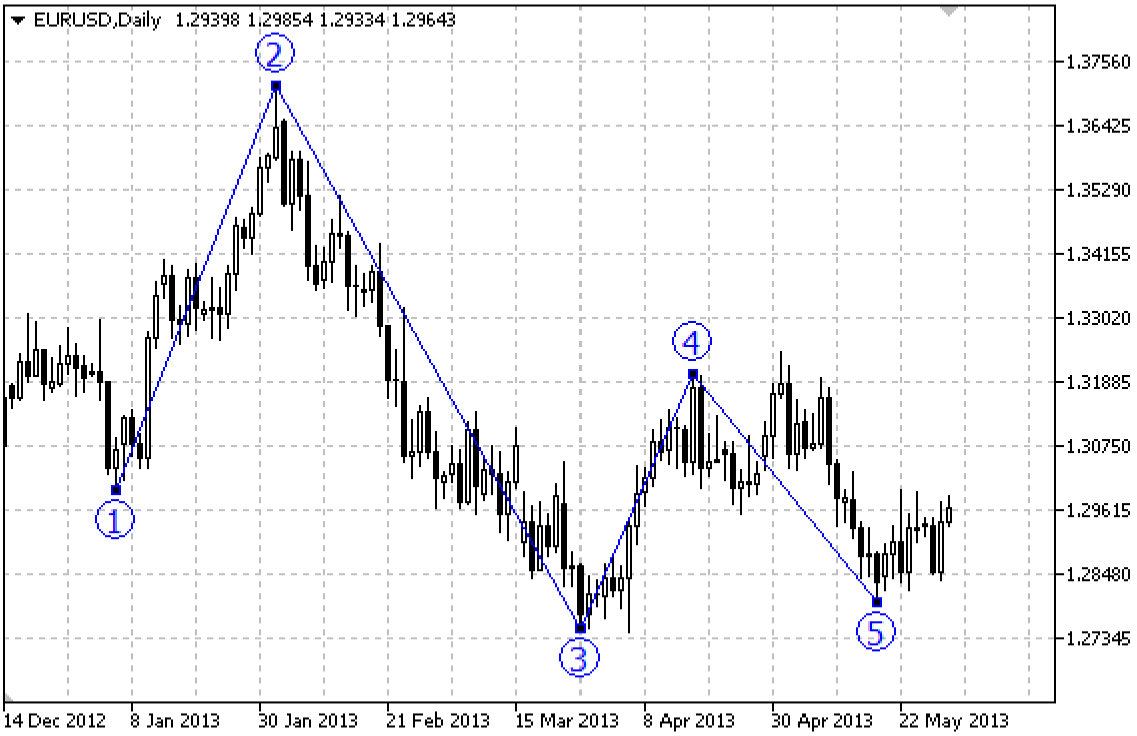
```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Expansion
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Expansion
   if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidder
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
```

```
          p1-=1;
       //--- move the point
       if(!FiboExpansionPointChange(0,InpName,0,date[d1],price[p1]))
          return;
       //--- check if the script's operation has been forcefully disabled
       if(IsStopped())
          return;
       //--- redraw the chart
       ChartRedraw();
      }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/2;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
      {
       //--- use the following value
       if(p3>1)
          p3-=1;
       //--- move the point
       if(!FiboExpansionPointChange(0,InpName,2,date[d3],price[p3]))
          return;
       //--- check if the script's operation has been forcefully disabled
       if(IsStopped())
          return;
       //--- redraw the chart
       ChartRedraw();
      }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
      {
       //--- use the following value
       if(p2<accuracy-1)
          p2+=1;
       //--- move the point
       if(!FiboExpansionPointChange(0,InpName,1,date[d2],price[p2]))
          return;
       //--- check if the script's operation has been forcefully disabled
       if(IsStopped())
          return;
       //--- redraw the chart
       ChartRedraw();
      }
//--- 1 second of delay
```

```
   Sleep(1000);
//--- delete the object from the chart
   FiboExpansionDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ELLIOTWAVE5

Elliott Motive Wave.



Note

For "Elliott Motive Wave", it is possible to enable/disable the mode of connecting points by lines (OBJPROP_DRAWLINES property), as well as set the level of wave positioning (from ENUM_ELLIOT_WAVE_DEGREE enumeration).

Example

The following script creates and moves Elliott motive wave on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Elliott Motive Wave\"."
#property description "Anchor point coordinates are set in percentage of the size of"
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string              InpName="ElliotWave5";    // Object name
input int                 InpDate1=10;              // 1 st point's date, %
input int                 InpPrice1=90;             // 1 st point's price, %
input int                 InpDate2=20;              // 2 nd point's date, %
input int                 InpPrice2=40;             // 2 nd point's price, %
input int                 InpDate3=30;              // 3 rd point's date, %
input int                     InpPrice3=60;            // 3 rd point's price, %
input int                     InpDate4=40;             // 4 th point's date, %
input int                     InpPrice4=10;            // 4 th point's price, %
input int                     InpDate5=60;             // 5 th point's date, %
input int                     InpPrice5=40;            // 5 th point's price, %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Level
input bool                    InpDrawLines=true;       // Displaying the lines
input color                   InpColor=clrRed;         // Color of the lines
input ENUM_LINE_STYLE         InpStyle=STYLE_DASH;     // Style of the lines
input int                     InpWidth=2;              // Width of the lines
input bool                    InpBack=false;           // Background object
input bool                    InpSelection=true;       // Highlight to move
input bool                    InpHidden=true;          // Hidden in the object list
input long                    InpZOrder=0;             // Priority for mouse click
//+------------------------------------------------------------------+
//| Create "Elliott Motive Wave" by the given coordinates            |
//+------------------------------------------------------------------+
bool ElliotWave5Create(const long                    chart_ID=0,              // chart
                       const string                  name="ElliotWave5",      // wave
                       const int                     sub_window=0,            // subw
                       datetime                      time1=0,                 // first
                       double                        price1=0,                // first
                       datetime                      time2=0,                 // seco
                       double                        price2=0,                // seco
                       datetime                      time3=0,                 // thir
                       double                        price3=0,                // thir
                       datetime                      time4=0,                 // fourt
                       double                        price4=0,                // fourt
                       datetime                      time5=0,                 // fift
                       double                        price5=0,                // fift
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // degre
                       const bool                    draw_lines=true,         // disp
                       const color                   clr=clrRed,              // obje
                       const ENUM_LINE_STYLE         style=STYLE_SOLID,       // style
                       const int                     width=1,                 // widt
                       const bool                    back=false,              // in t
                       const bool                    selection=true,          // high
                       const bool                    hidden=true,             // hidd
                       const long                    z_order=0)               // prio
  {
//--- set anchor points' coordinates if they are not set
   ChangeElliotWave5EmptyPoints(time1,price1,time2,price2,time3,price3,time4,price4,t
//--- reset the error value
   ResetLastError();
//--- Create "Elliott Motive Wave" by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE5,sub_window,time1,price1,time2,price
      price3,time4,price4,time5,price5))
     {
      Print(__FUNCTION__,
            ": failed to create \"Elliott Motive Wave\"! Error code = ",GetLastError()
      return(false);
     }
//--- set degree (wave size)
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- enable (true) or disable (false) the mode of displaying the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- set the object's color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for moving
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move anchor point of Elliott Motive Wave                         |
//+------------------------------------------------------------------+
bool ElliotWave5PointChange(const long   chart_ID=0,         // chart's ID
                            const string name="ElliotWave5", // object name
                            const int    point_index=0,      // anchor point index
                            datetime     time=0,             // anchor point time coor
                            double       price=0)            // anchor point price coo
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Elliott Motive Wave                                       |
//+------------------------------------------------------------------+
bool ElliotWave5Delete(const long   chart_ID=0,         // chart's ID
                       const string name="ElliotWave5") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Elliott Motive Wave\"! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Elliott Motive Wave's anchor points and      |
//| set default values for empty ones                                |
//+------------------------------------------------------------------+
void ChangeElliotWave5EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3,
                                  datetime &time4,double &price4,
                                  datetime &time5,double &price5)
  {
//--- array for receiving the open time of the last 10 bars
   datetime temp[];
   ArrayResize(temp,10);
//--- receive data
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- receive the value of one point on the current chart
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the first point's time is not set, it will be 9 bars left from the last bar
   if(!time1)
      time1=temp[0];
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it will be 7 bars left from the last bar
   if(!time2)
      time2=temp[2];
```

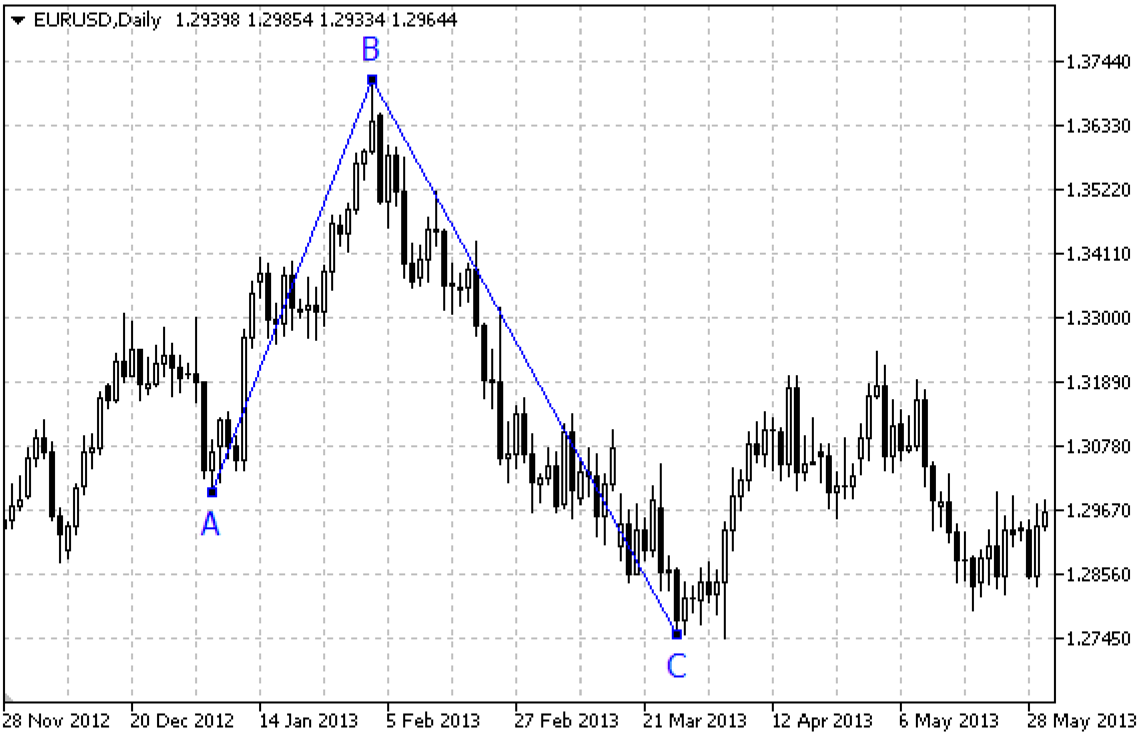
```
//--- if the second point's price is not set, move it 300 points lower than the first
   if(!price2)
      price2=price1-300*point;
//--- if the third point's time is not set, it will be 5 bars left from the last bar
   if(!time3)
      time3=temp[4];
//--- if the third point's price is not set, move it 250 points lower than the first c
   if(!price3)
      price3=price1-250*point;
//--- if the fourth point's time is not set, it will be 3 bars left from the last bar
   if(!time4)
      time4=temp[6];
//--- if the fourth point's price is not set, move it 550 points lower than the first
   if(!price4)
      price4=price1-550*point;
//--- if the fifth point's time is not set, it will be on the last bar
   if(!time5)
      time5=temp[9];
//--- if the fifth point's price is not set, move it 450 points lower than the first c
   if(!price5)
      price5=price1-450*point;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100 ||
      InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
      InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
```

```
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Elliott Motive Wave
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int d4=InpDate4*(bars-1)/100;
   int d5=InpDate5*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
   int p4=InpPrice4*(accuracy-1)/100;
   int p5=InpPrice5*(accuracy-1)/100;
//--- Create Elliott Motive Wave
   if(!ElliotWave5Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pr
      date[d4],price[p4],date[d5],price[p5],InpDegree,InpDrawLines,InpColor,InpStyle,I
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the fifth anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p5<accuracy-1)
         p5+=1;
      //--- move the point
      if(!ElliotWave5PointChange(0,InpName,4,date[d5],price[p5]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
```

```
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/5;
//--- move the second and third anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p2<accuracy-1)
         p2+=1;
      if(p3>1)
         p3-=1;
      //--- shift the points
      if(!ElliotWave5PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      if(!ElliotWave5PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the first and fourth anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1>1)
         p1-=1;
      if(p4<accuracy-1)
         p4+=1;
      //--- shift the points
      if(!ElliotWave5PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave5PointChange(0,InpName,3,date[d4],price[p4]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   ElliotWave5Delete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ELLIOTWAVE3

Elliott Correction Wave.



Note

For "Elliott Correction Wave", it is possible to enable/disable the mode of connecting points by lines (OBJPROP_DRAWLINES property), as well as set the level of wave positioning (from ENUM_ELLIOT_WAVE_DEGREE enumeration).

Example

The following script creates and moves Elliott correction wave on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

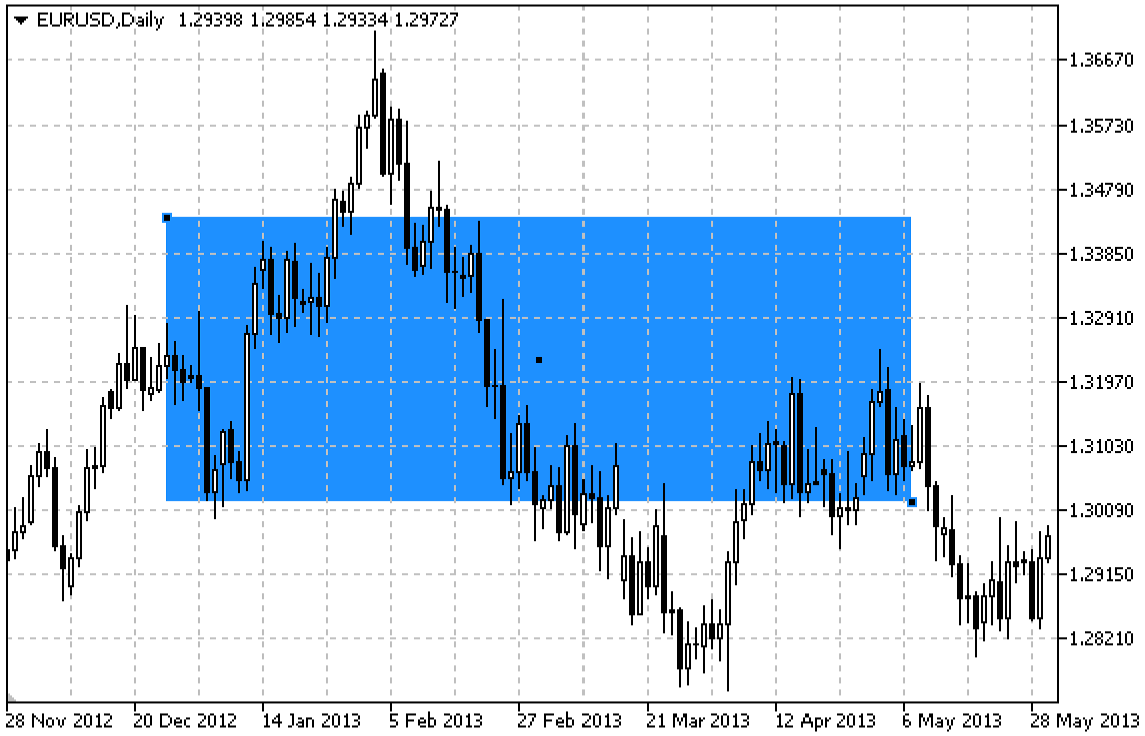
```
//--- description
#property description "Script draws \"Elliott Correction Wave\" graphical object."
#property description "Anchor point coordinates are set in percentage of the chart's
#property description "size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string              InpName="ElliotWave3";    // Object name
input int                 InpDate1=10;              // 1 st point's date, %
input int                 InpPrice1=90;             // 1 st point's price, %
input int                 InpDate2=30;              // 2 nd point's date, %
input int                 InpPrice2=10;             // 2 nd point's price, %
input int                 InpDate3=50;              // 3 rd point's date, %
input int                     InpPrice3=40;            // 3 rd point's price, %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Level
input bool                    InpDrawLines=true;       // Displaying the lines
input color                   InpColor=clrRed;         // Color of the lines
input ENUM_LINE_STYLE         InpStyle=STYLE_DASH;     // Style of the lines
input int                     InpWidth=2;              // Width of the lines
input bool                    InpBack=false;           // Background object
input bool                    InpSelection=true;       // Highlight to move
input bool                    InpHidden=true;          // Hidden in the object list
input long                    InpZOrder=0;             // Priority for mouse click
//+------------------------------------------------------------------+
//| Create "Elliott Correction Wave" by the given coordinates        |
//+------------------------------------------------------------------+
bool ElliotWave3Create(const long                    chart_ID=0,              // chart
                       const string                  name="ElliotWave3",      // wave
                       const int                     sub_window=0,            // subw
                       datetime                      time1=0,                 // first
                       double                        price1=0,                // first
                       datetime                      time2=0,                 // seco
                       double                        price2=0,                // seco
                       datetime                      time3=0,                 // thir
                       double                        price3=0,                // thir
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // degr
                       const bool                    draw_lines=true,         // disp
                       const color                   clr=clrRed,              // obje
                       const ENUM_LINE_STYLE         style=STYLE_SOLID,       // styl
                       const int                     width=1,                 // widt
                       const bool                    back=false,              // in t
                       const bool                    selection=true,          // high
                       const bool                    hidden=true,             // hidd
                       const long                    z_order=0)               // prio
  {
//--- set anchor points' coordinates if they are not set
   ChangeElliotWave3EmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- Create "Elliott Correction Wave" by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE3,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": failed to create \"Elliott Correction Wave\"! Error code = ",GetLastEr
      return(false);
     }
//--- set degree (wave size)
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- enable (true) or disable (false) the mode of displaying the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- set the object's color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
```

```
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channel for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move anchor point of Elliott Correction Wave                      |
//+------------------------------------------------------------------+
bool ElliotWave3PointChange(const long   chart_ID=0,         // chart's ID
                            const string name="ElliotWave3", // object name
                            const int    point_index=0,      // anchor point index
                            datetime     time=0,             // anchor point time coor
                            double       price=0)            // anchor point price coo
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Elliott Correction Wave                                    |
//+------------------------------------------------------------------+
bool ElliotWave3Delete(const long   chart_ID=0,         // chart's ID
                       const string name="ElliotWave3") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Elliott Correction Wave\"! Error code = ",GetLastEr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Elliott Correction Wave's anchor points      |
//| and set default values for empty ones                            |
//+------------------------------------------------------------------+
void ChangeElliotWave3EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3)
  {
//--- array for receiving the open time of the last 10 bars
   datetime temp[];
   ArrayResize(temp,10);
//--- receive data
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- receive the value of one point on the current chart
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the first point's time is not set, it will be 9 bars left from the last bar
   if(!time1)
      time1=temp[0];
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it will be 5 bars left from the last bar
   if(!time2)
      time2=temp[4];
//--- if the second point's price is not set, move it 300 points lower than the first
   if(!price2)
      price2=price1-300*point;
//--- if the third point's time is not set, it will be 1 bar left from the last bar
   if(!time3)
      time3=temp[8];
//--- if the third point's price is not set, move it 200 points lower than the first
   if(!price3)
      price3=price1-200*point;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Elliott Correction Wave
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- Create Elliott Correction Wave
   if(!ElliotWave3Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pr
      InpDegree,InpDrawLines,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3<accuracy-1)
         p3+=1;
      //--- move the point
      if(!ElliotWave3PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the first and second anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1>1)
         p1-=1;
      if(p2<accuracy-1)
         p2+=1;
      //--- shift the points
      if(!ElliotWave3PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave3PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
```

```
   ElliotWave3Delete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE

Rectangle.



Note

For rectangle, the mode of filling with color can be set using OBJPROP_FILL property.

Example

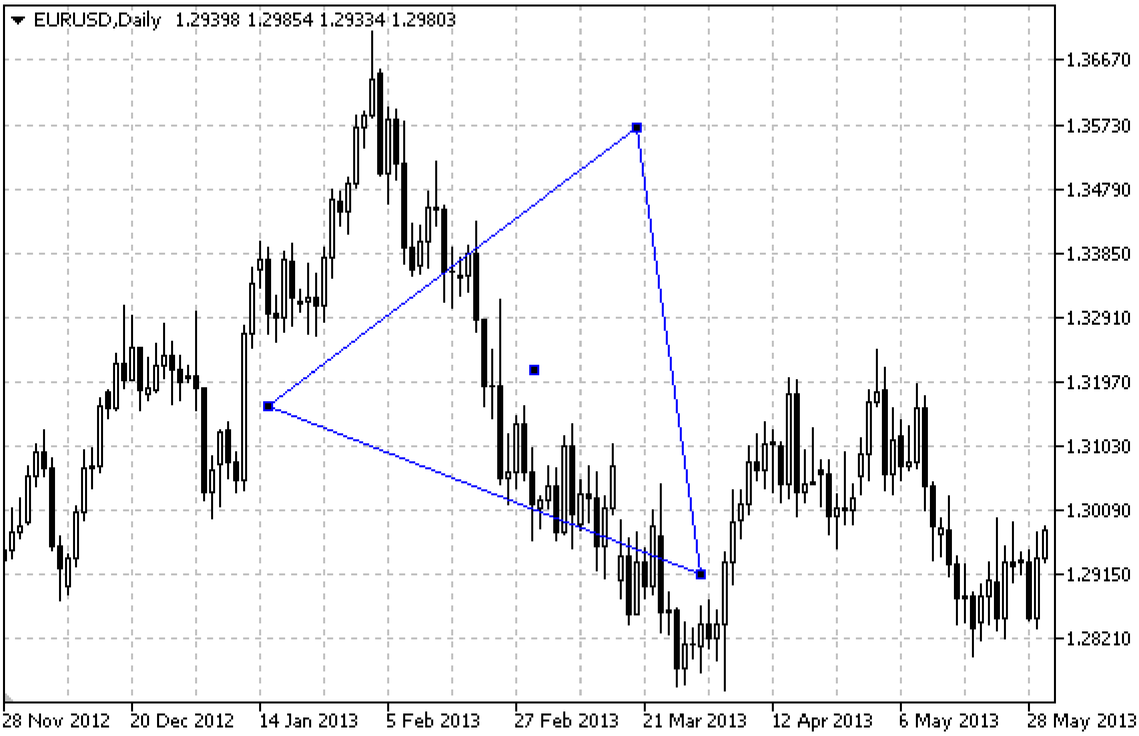
The following script creates and moves the rectangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates rectangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Rectangle"; // Rectangle name
input int             InpDate1=40;         // 1 st point's date, %
input int             InpPrice1=40;        // 1 st point's price, %
input int             InpDate2=60;         // 2 nd point's date, %
input int             InpPrice2=60;        // 2 nd point's price, %
input color           InpColor=clrRed;     // Rectangle color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of rectangle lines
input int             InpWidth=2;          // Width of rectangle lines
input bool            InpFill=true;        // Filling the rectangle with color
input bool            InpBack=false;       // Background rectangle
input bool            InpSelection=true;   // Highlight to move
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create rectangle by the given coordinates                        |
//+------------------------------------------------------------------+
bool RectangleCreate(const long            chart_ID=0,        // chart's ID
                     const string          name="Rectangle",  // rectangle name
                     const int             sub_window=0,      // subwindow index
                     datetime              time1=0,           // first point time
                     double                price1=0,          // first point price
                     datetime              time2=0,           // second point time
                     double                price2=0,          // second point price
                     const color           clr=clrRed,        // rectangle color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // style of rectangle l
                     const int             width=1,           // width of rectangle l
                     const bool            fill=false,        // filling rectangle wit
                     const bool            back=false,        // in the background
                     const bool            selection=true,    // highlight to move
                     const bool            hidden=true,       // hidden in the object
                     const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a rectangle by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle! Error code = ",GetLastError());
      return(false);
     }
//--- set rectangle color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the style of rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the rectangle
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the rectangle for mov
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the rectangle anchor point                                  |
//+------------------------------------------------------------------+
bool RectanglePointChange(const long   chart_ID=0,       // chart's ID
                          const string name="Rectangle", // rectangle name
                          const int    point_index=0,    // anchor point index
                          datetime     time=0,           // anchor point time coordina
                          double       price=0)          // anchor point price coordir
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the rectangle                                             |
//+------------------------------------------------------------------+
bool RectangleDelete(const long   chart_ID=0,       // chart's ID
                     const string name="Rectangle") // rectangle name
  {
//--- reset the error value
   ResetLastError();
//--- delete rectangle
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete rectangle! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of rectangle's anchor points and set default    |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower than the first
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing rectangle anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the rectangle
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a rectangle
   if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the rectangle's anchor points
//--- loop counter
   int h_steps=bars/2;
//--- move the anchor points
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- shift the points
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- shift the points
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the rectangle from the chart
   RectangleDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TRIANGLE

Triangle.



Note

For triangle, the mode of filling with color can be set using OBJPROP_FILL property.

Example

The following script creates and moves the triangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates triangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Triangle";        // Triangle name
input int             InpDate1=25;               // 1 st point's date, %
input int             InpPrice1=50;              // 1 st point's price, %
input int             InpDate2=70;               // 2 nd point's date, %
input int             InpPrice2=70;              // 2 nd point's price, %
input int             InpDate3=65;               // 3 rd point's date, %
input int             InpPrice3=20;              // 3 rd point's price, %
input color           InpColor=clrRed;           // Triangle color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of triangle lines
input int             InpWidth=2;                // Width of triangle lines
input bool            InpFill=false;             // Filling triangle with color
input bool            InpBack=false;             // Background triangle
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object list
input long            InpZOrder=0;               // Priority for mouse click
//+------------------------------------------------------------------+
//| Create triangle by the given coordinates                         |
//+------------------------------------------------------------------+
bool TriangleCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="Triangle",   // triangle name
                    const int             sub_window=0,      // subwindow index
                    datetime              time1=0,           // first point time
                    double                price1=0,          // first point price
                    datetime              time2=0,           // second point time
                    double                price2=0,          // second point price
                    datetime              time3=0,           // third point time
                    double                price3=0,          // third point price
                    const color           clr=clrRed,        // triangle color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of triangle line
                    const int             width=1,           // width of triangle line
                    const bool            fill=false,        // filling triangle with
                    const bool            back=false,        // in the background
                    const bool            selection=true,    // highlight to move
                    const bool            hidden=true,       // hidden in the object l
                    const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor points' coordinates if they are not set
   ChangeTriangleEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create triangle by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": failed to create a triangle! Error code = ",GetLastError());
      return(false);
     }
//--- set triangle color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of triangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of triangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the triangle
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of highlighting the triangle for movin
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the triangle anchor point                                   |
//+------------------------------------------------------------------+
bool TrianglePointChange(const long   chart_ID=0,      // chart's ID
                         const string name="Triangle", // triangle name
                         const int    point_index=0,   // anchor point index
                         datetime     time=0,          // anchor point time coordinate
                         double       price=0)         // anchor point price coordinat
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the triangle                                              |
//+------------------------------------------------------------------+
bool TriangleDelete(const long   chart_ID=0,      // chart's ID
                    const string name="Triangle") // triangle name
  {
//--- reset the error value
   ResetLastError();
//--- delete the triangle
   if(!ObjectDelete(chart_ID,name))
```

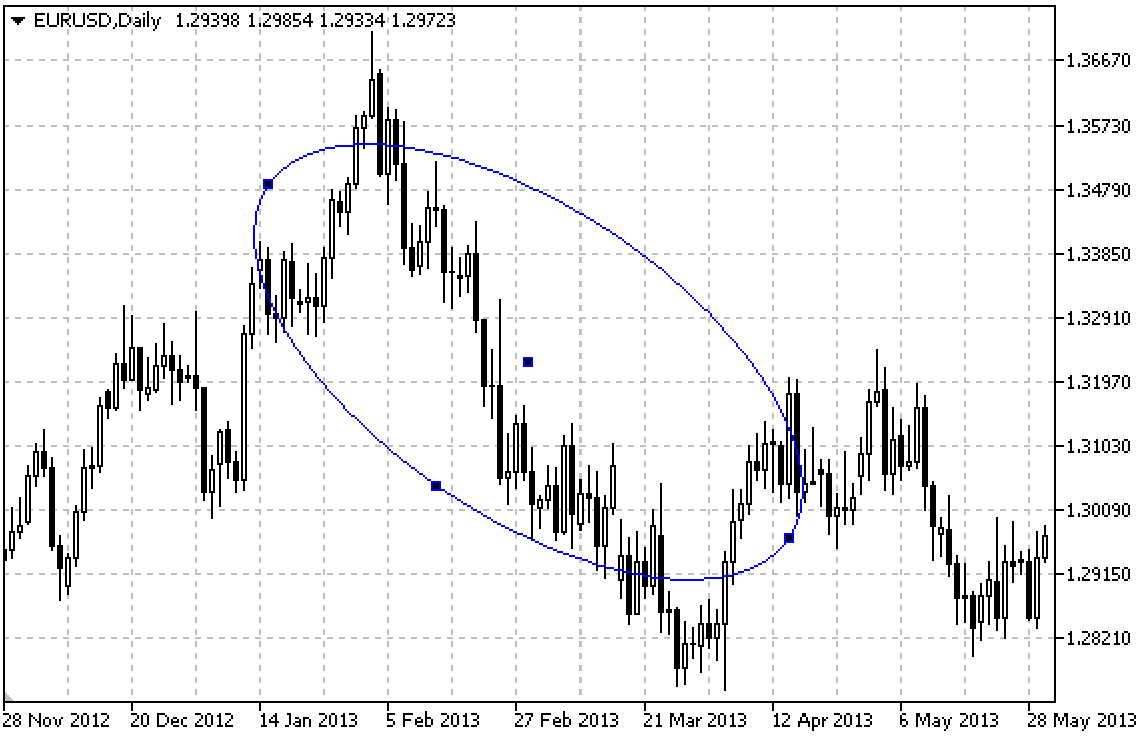
```
     {
      Print(__FUNCTION__,
            ": failed to delete the ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of triangle's anchor points and set default     |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeTriangleEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2,
                               datetime &time3,double &price3)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower than the first
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second point's date
   if(!time3)
      time3=time2;
//--- if the third point's price is not set, it is equal to the first point's one
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing triangle anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the triangle
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create a triangle
   if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the triangle anchor points
//--- loop counter
```

```
   int v_steps=accuracy*3/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars*9/20-1;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2>1)
         d2-=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/4;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3<accuracy-1)
         p3+=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))
```

```
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete triangle from the chart
   TriangleDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ELLIPSE

Ellipse.



Note

For ellipse, the mode of filling with color can be set using OBJPROP_FILL property.

Example

The following script creates and moves the ellipse on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

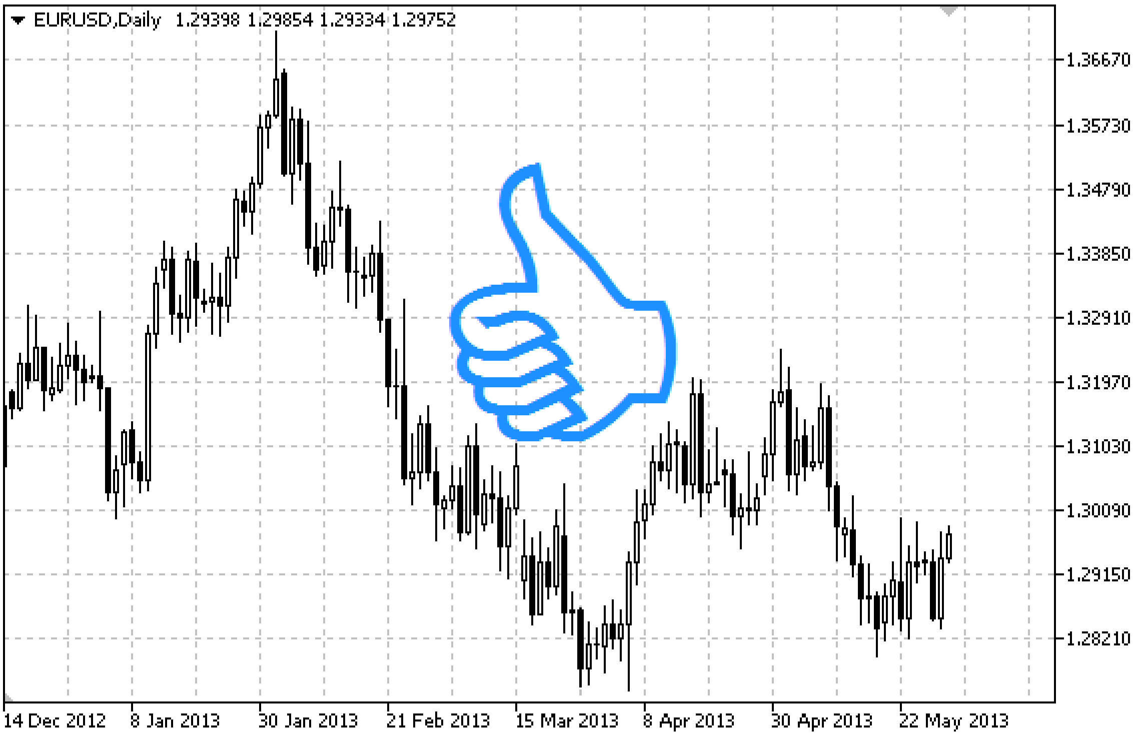
```
//--- description
#property description "Script creates ellipse on the chart."
#property description "Anchor point coordinates are set"
#property description "in percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Ellipse";          // Ellipse name
input int             InpDate1=30;                // 1 st point's date, %
input int             InpPrice1=20;               // 1 st point's price, %
input int             InpDate2=70;                // 2 nd point's date, %
input int             InpPrice2=80;               // 2 nd point's price, %
input int             InpDate3=50;                // 3 rd point's date, %
input int             InpPrice3=60;               // 3 rd point's price, %
input color           InpColor=clrRed;            // Ellipse color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT;  // Style of ellipse lines
input int             InpWidth=2;              // Width of ellipse lines
input bool            InpFill=false;           // Filling ellipse with color
input bool            InpBack=false;           // Background ellipse
input bool            InpSelection=true;       // Highlight to move
input bool            InpHidden=true;          // Hidden in the object list
input long            InpZOrder=0;             // Priority for mouse click
//+------------------------------------------------------------------+
//| Create an ellipse by the given coordinates                       |
//+------------------------------------------------------------------+
bool EllipseCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="Ellipse",    // ellipse name
                   const int             sub_window=0,      // subwindow index
                   datetime              time1=0,           // first point time
                   double                price1=0,          // first point price
                   datetime              time2=0,           // second point time
                   double                price2=0,          // second point price
                   datetime              time3=0,           // third point time
                   double                price3=0,          // third point price
                   const color           clr=clrRed,        // ellipse color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of ellipse lines
                   const int             width=1,           // width of ellipse lines
                   const bool            fill=false,        // filling ellipse with co
                   const bool            back=false,        // in the background
                   const bool            selection=true,    // highlight to move
                   const bool            hidden=true,       // hidden in the object li
                   const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor points' coordinates if they are not set
   ChangeEllipseEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create an ellipse by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time2,price2,tir
     {
      Print(__FUNCTION__,
            ": failed to create an ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- set an ellipse color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of ellipse lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of ellipse lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- enable (true) or disable (false) the mode of filling the ellipse
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the ellipse for movin
```

```
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the ellipse anchor point                                    |
//+------------------------------------------------------------------+
bool EllipsePointChange(const long   chart_ID=0,      // chart's ID
                        const string name="Ellipse", // ellipse name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time coordinate
                        double       price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete ellipse                                                   |
//+------------------------------------------------------------------+
bool EllipseDelete(const long   chart_ID=0,      // chart's ID
                   const string name="Ellipse") // ellipse name
  {
//--- reset the error value
   ResetLastError();
//--- delete an ellipse
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete an ellipse! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of ellipse anchor points and set default values |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2,
                              datetime &time3,double &price3)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left from the secon
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower than the first
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second point's date
   if(!time3)
      time3=time2;
//--- if the third point's price is not set, it is equal to the first point's one
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing ellipse anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the ellipse
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create an ellipse
   if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the ellipse anchor points
//--- loop counter
   int v_steps=accuracy/5;
```

```
//--- move the first and second anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- shift the points
      if(!EllipsePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!EllipsePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/5;
//--- move the third anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d3>1)
         d3-=1;
      //--- move the point
      if(!EllipsePointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete ellipse from the chart
   EllipseDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_THUMB_UP

Thumbs Up sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

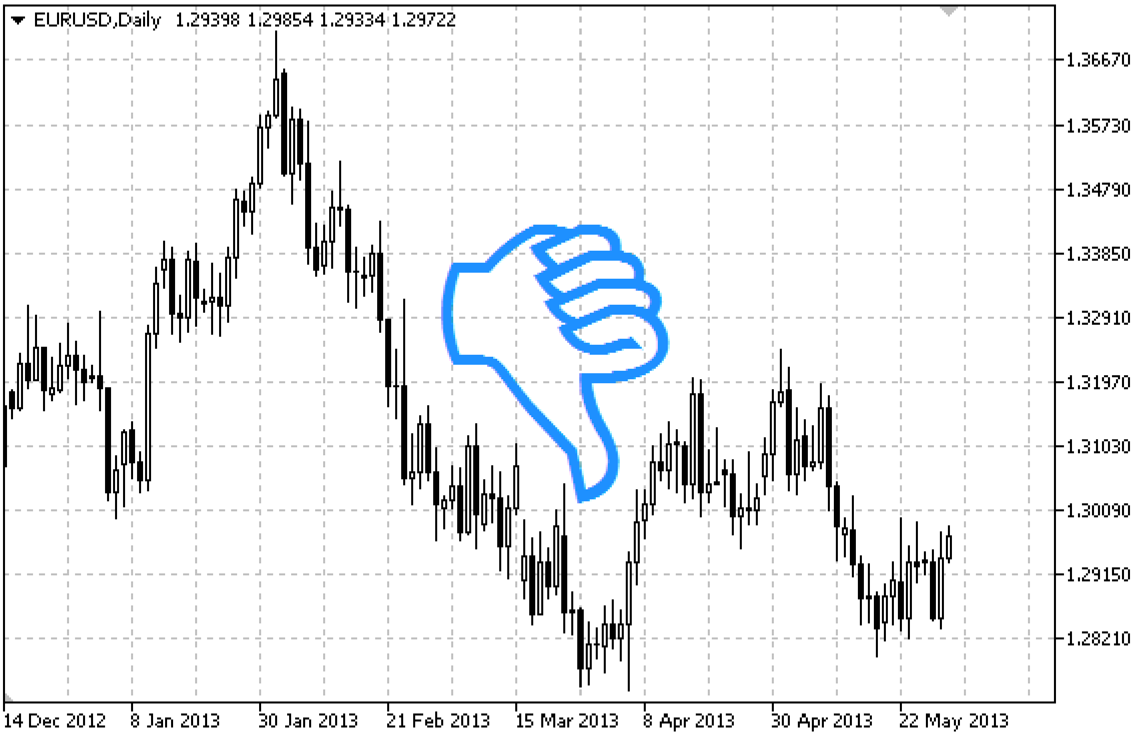
```
//--- description
#property description "Script draws \"Thumbs Up\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ThumbUp";     // Sign name
input int               InpDate=75;            // Anchor point date in %
input int               InpPrice=25;           // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP;  // Anchor type
input color             InpColor=clrRed;       // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;    // Border line style
input int               InpWidth=5;            // Sign size
input bool              InpBack=false;         // Background sign
input bool              InpSelection=true;     // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;           // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Thumbs Up sign                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpCreate(const long              chart_ID=0,           // chart's ID
                        const string            name="ThumbUp",       // sign name
                        const int               sub_window=0,         // subwindow in
                        datetime                time=0,               // anchor point
                        double                  price=0,              // anchor point
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                        const color             clr=clrRed,           // sign color
                        const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line
                        const int               width=3,              // sign size
                        const bool              back=false,           // in the backg
                        const bool              selection=true,       // highlight to
                        const bool              hidden=true,          // hidden in the
                        const long              z_order=0)            // priority for
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpMove(const long   chart_ID=0,     // chart's ID
                      const string name="ThumbUp", // object name
                      datetime     time=0,         // anchor point time coordinate
                      double       price=0)        // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Thumbs Up sign anchor type                                |
//+------------------------------------------------------------------+
bool ArrowThumbUpAnchorChange(const long              chart_ID=0,        // chart's I
                              const string            name="ThumbUp",    // object na
                              const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor typ
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
```

```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Thumbs Up sign                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpDelete(const long   chart_ID=0,     // chart's ID
                        const string name="ThumbUp") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
```

```
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Thumbs Up sign on the chart
   if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int h_steps=bars/4;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d>1)
         d-=1;
      //--- move the point
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/4;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- change anchor point location relative to the sign
   ArrowThumbUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowThumbUpDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_THUMB_DOWN

Thumbs Down sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Thumbs Down\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ThumbDown";     // Sign name
input int               InpDate=25;              // Anchor point date in %
input int               InpPrice=75;             // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;        // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;     // Border line style
input int               InpWidth=5;             // Sign size
input bool              InpBack=false;          // Background sign
input bool              InpSelection=true;      // Highlight to move
input bool              InpHidden=true;         // Hidden in the object list
input long              InpZOrder=0;            // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Thumbs Down sign                                           |
//+------------------------------------------------------------------+
bool ArrowThumbDownCreate(const long              chart_ID=0,           // chart's ID
                          const string            name="ThumbDown",     // sign name
                          const int               sub_window=0,         // subwindow
                          datetime                time=0,               // anchor poi
                          double                  price=0,              // anchor poi
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor typ
                          const color             clr=clrRed,           // sign color
                          const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border lin
                          const int               width=3,              // sign size
                          const bool              back=false,           // in the bac
                          const bool              selection=true,       // highlight
                          const bool              hidden=true,          // hidden in
                          const long              z_order=0)            // priority f
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
```

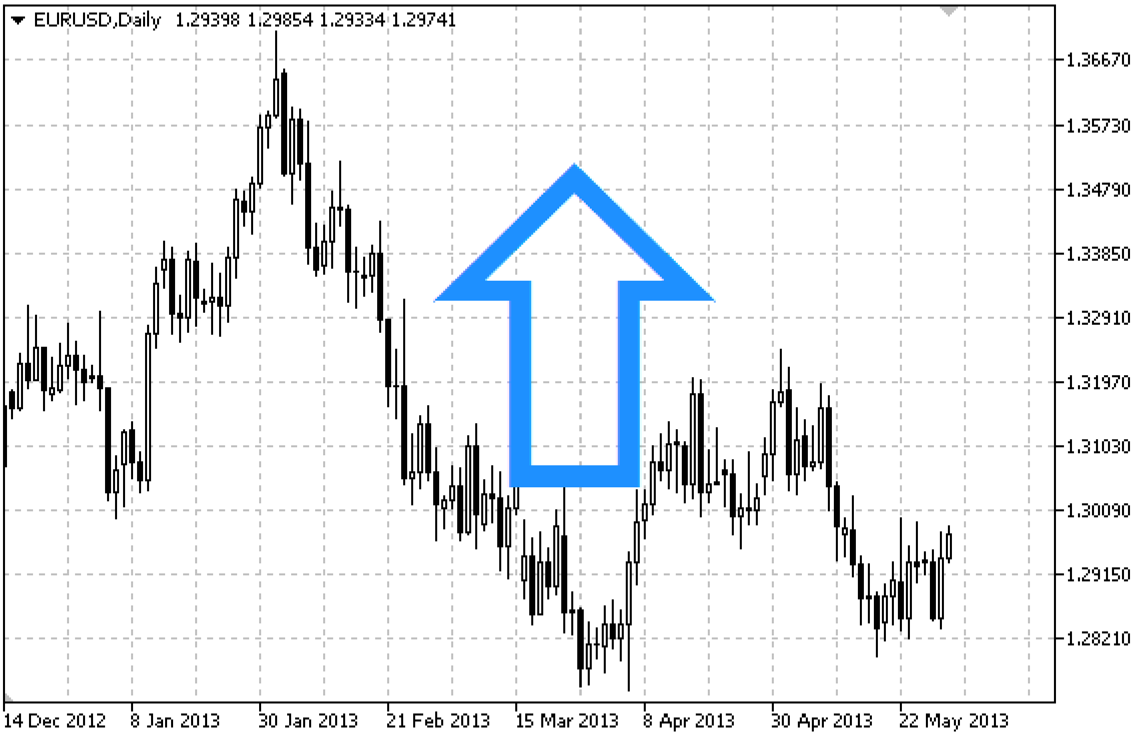
```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowThumbDownMove(const long   chart_ID=0,       // chart's ID
                        const string name="ThumbDown", // object name
                        datetime     time=0,           // anchor point time coordinate
                        double       price=0)          // anchor point price coordinat
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Thumbs Down sign anchor type                              |
//+------------------------------------------------------------------+
bool ArrowThumbDownAnchorChange(const long              chart_ID=0,        // chart's
                                const string            name="ThumbDown",  // object 
                                const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor 
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Thumbs Down sign                                          |
//+------------------------------------------------------------------+
bool ArrowThumbDownDelete(const long   chart_ID=0,       // chart's ID
                          const string name="ThumbDown") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
```

```
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Thumbs Down sign on the chart
   if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int h_steps=bars/4;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
```

```
            return;
         //--- redraw the chart
         ChartRedraw();
         // 0.05 seconds of delay
         Sleep(50);
        }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/4;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- change anchor point location relative to the sign
   ArrowThumbDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowThumbDownDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_UP

Arrow Up sign.


EURUSD,Daily 1.29398 1.29854 1.29334 1.29741
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
14 Dec 2012
8 Jan 2013
30 Jan 2013
21 Feb 2013
15 Mar 2013
8 Apr 2013
30 Apr 2013
22 May 2013

Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

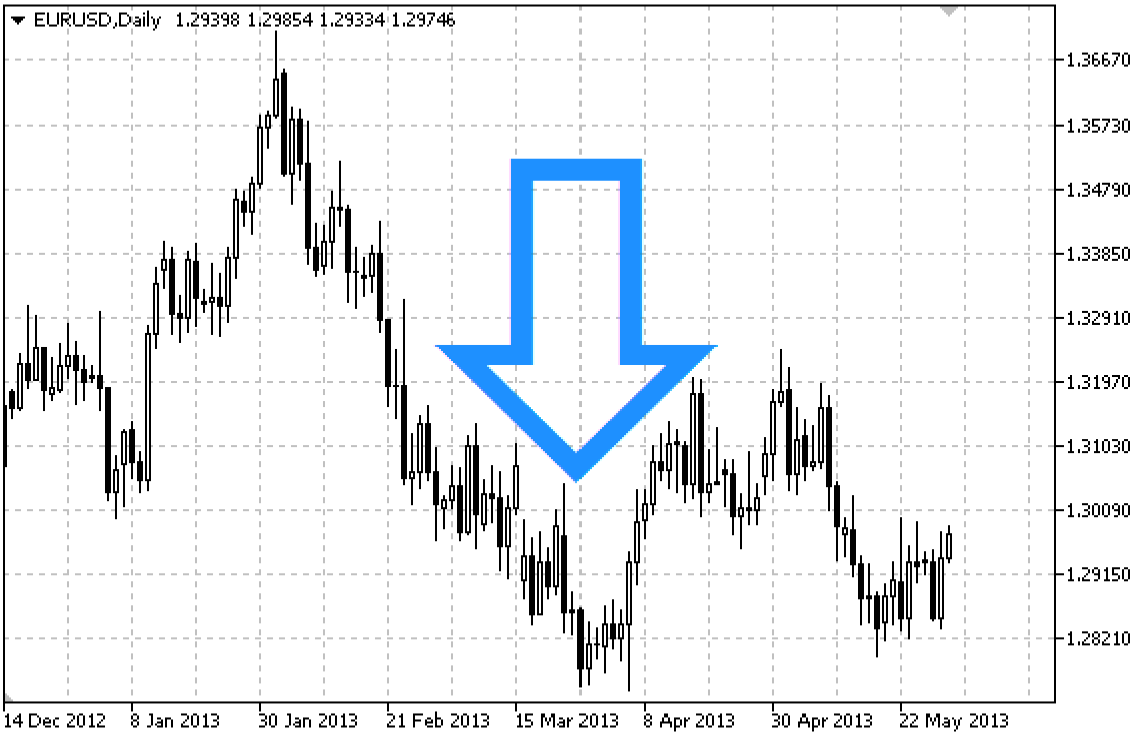
```
//--- description
#property description "Script draws \"Arrow Up\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowUp";     // Sign name
input int               InpDate=25;            // Anchor point date in %
input int               InpPrice=25;           // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color             InpColor=clrRed;       // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Border line style
input int               InpWidth=5;           // Sign size
input bool              InpBack=false;        // Background sign
input bool              InpSelection=false;   // Highlight to move
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+-------------------------------------------------------------------+
//| Create Arrow Up sign                                              |
//+-------------------------------------------------------------------+
bool ArrowUpCreate(const long              chart_ID=0,           // chart's ID
                   const string            name="ArrowUp",       // sign name
                   const int               sub_window=0,         // subwindow index
                   datetime                time=0,               // anchor point time
                   double                  price=0,              // anchor point pric
                   const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                   const color             clr=clrRed,           // sign color
                   const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line style
                   const int               width=3,              // sign size
                   const bool              back=false,           // in the background
                   const bool              selection=true,       // highlight to move
                   const bool              hidden=true,          // hidden in the obje
                   const long              z_order=0)            // priority for mouse
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowUpMove(const long   chart_ID=0,      // chart's ID
                 const string name="ArrowUp", // object name
                 datetime     time=0,          // anchor point time coordinate
                 double       price=0)         // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Arrow Down sign anchor type                               |
//+------------------------------------------------------------------+
bool ArrowUpAnchorChange(const long              chart_ID=0,        // chart's ID
                         const string            name="ArrowUp",    // object name
                         const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
  {
//--- reset the error value
   ResetLastError();
//--- change anchor point location
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
```

```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Arrow Up sign                                             |
//+------------------------------------------------------------------+
bool ArrowUpDelete(const long   chart_ID=0,     // chart's ID
                   const string name="ArrowUp") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Up\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
```

```
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Up sign on the chart
   if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change anchor point location relative to the sign
   ArrowUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowUpDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_DOWN

Arrow Down sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Arrow Down\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowDown";      // Sign name
input int               InpDate=75;               // Anchor point date in %
input int               InpPrice=75;              // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;          // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;       // Border line style
input int               InpWidth=5;              // Sign size
input bool              InpBack=false;           // Background sign
input bool              InpSelection=false;      // Highlight to move
input bool              InpHidden=true;          // Hidden in the object list
input long              InpZOrder=0;             // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Arrow Down sign                                           |
//+------------------------------------------------------------------+
bool ArrowDownCreate(const long              chart_ID=0,           // chart's ID
                     const string            name="ArrowDown",     // sign name
                     const int               sub_window=0,         // subwindow index
                     datetime                time=0,               // anchor point ti
                     double                  price=0,              // anchor point pr
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                     const color             clr=clrRed,           // sign color
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line sty
                     const int               width=3,              // sign size
                     const bool              back=false,           // in the backgrou
                     const bool              selection=true,       // highlight to mo
                     const bool              hidden=true,          // hidden in the o
                     const long              z_order=0)            // priority for mo
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
```

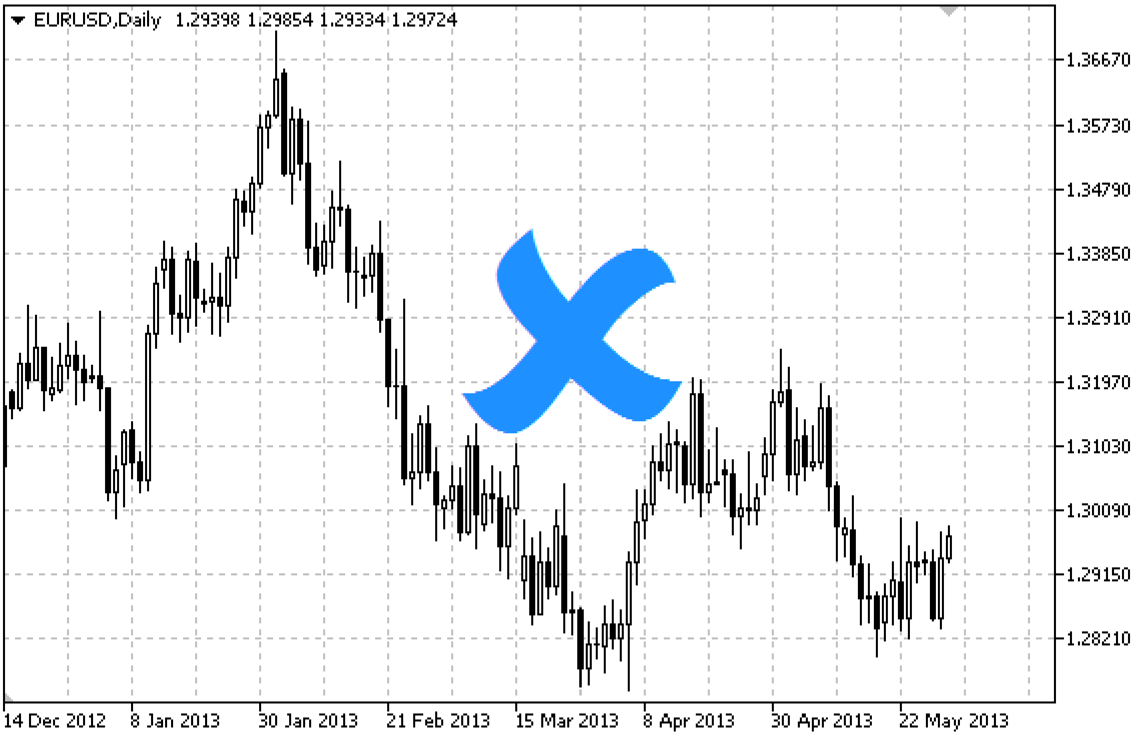
```
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowDownMove(const long   chart_ID=0,        // chart's ID
                   const string name="ArrowDown", // object name
                   datetime     time=0,            // anchor point time coordinate
                   double       price=0)           // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Arrow Down sign anchor type                               |
//+------------------------------------------------------------------+
bool ArrowDownAnchorChange(const long              chart_ID=0,        // chart's ID
                           const string            name="ArrowDown",  // object name
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
  {
//--- reset the error value
   ResetLastError();
//--- change anchor point location
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Arrow Down sign                                           |
//+------------------------------------------------------------------+
bool ArrowDownDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowDown") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Down\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
```

```
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Down sign on the chart
   if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change anchor point location relative to the sign
   ArrowDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowDownDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_STOP

Stop sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Stop sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Stop\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowStop";     // Sign name
input int               InpDate=10;              // Anchor point date in %
input int               InpPrice=50;             // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;         // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Border line style
input int               InpWidth=5;              // Sign size
input bool              InpBack=false;           // Background sign
input bool              InpSelection=false;      // Highlight to move
input bool              InpHidden=true;          // Hidden in the object list
input long              InpZOrder=0;             // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Stop sign                                                 |
//+------------------------------------------------------------------+
bool ArrowStopCreate(const long              chart_ID=0,           // chart's ID
                     const string            name="ArrowStop",     // sign name
                     const int               sub_window=0,         // subwindow index
                     datetime                time=0,               // anchor point ti
                     double                  price=0,              // anchor point pr
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                     const color             clr=clrRed,           // sign color
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line sty
                     const int               width=3,              // sign size
                     const bool              back=false,           // in the backgrou
                     const bool              selection=true,       // highlight to mo
                     const bool              hidden=true,          // hidden in the o
                     const long              z_order=0)            // priority for mou
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Stop\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowStopMove(const long   chart_ID=0,         // chart's ID
                   const string name="ArrowStop", // object name
                   datetime     time=0,            // anchor point time coordinate
                   double       price=0)           // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Stop sign anchor type                                     |
//+------------------------------------------------------------------+
bool ArrowStopAnchorChange(const long              chart_ID=0,         // chart's ID
                           const string            name="ArrowStop",   // object name
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor point
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
```

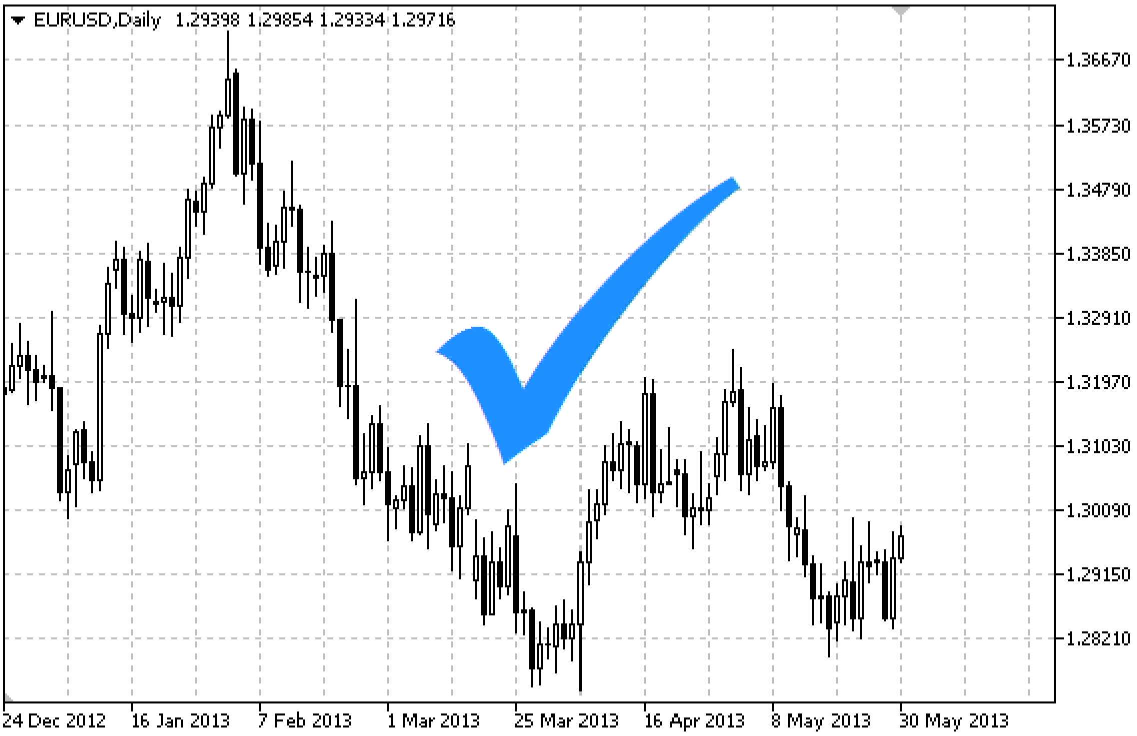
```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Stop sign                                                 |
//+------------------------------------------------------------------+
bool ArrowStopDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowStop") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Stop\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
```

```
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Stop sign on the chart
   if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
```

```
       //--- redraw the chart
       ChartRedraw();
       // 0.025 seconds of delay
       Sleep(25);
      }
//--- change anchor point location relative to the sign
   ArrowStopAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- loop counter
   h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.025 seconds of delay
      Sleep(25);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowStopDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_CHECK

Check sign.


EURUSD,Daily 1.29398 1.29854 1.29334 1.29716
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Check sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Check\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowCheck"; // Sign name
input int               InpDate=10;           // Anchor point date in %
input int               InpPrice=50;          // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color             InpColor=clrRed;      // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;    // Border line style
input int               InpWidth=5;            // Sign size
input bool              InpBack=false;         // Background sign
input bool              InpSelection=false;    // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;           // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Check sign                                                |
//+------------------------------------------------------------------+
bool ArrowCheckCreate(const long              chart_ID=0,           // chart's ID
                      const string            name="ArrowCheck",    // sign name
                      const int               sub_window=0,         // subwindow inde
                      datetime                time=0,               // anchor point t
                      double                  price=0,              // anchor point p
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor type
                      const color             clr=clrRed,           // sign color
                      const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line sty
                      const int               width=3,              // sign size
                      const bool              back=false,           // in the backgrou
                      const bool              selection=true,       // highlight to mo
                      const bool              hidden=true,          // hidden in the o
                      const long              z_order=0)            // priority for mo
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Check\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowCheckMove(const long   chart_ID=0,         // chart's ID
                    const string name="ArrowCheck", // object name
                    datetime     time=0,            // anchor point time coordinate
                    double       price=0)           // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Check anchor type                                         |
//+------------------------------------------------------------------+
bool ArrowCheckAnchorChange(const long              chart_ID=0,         // chart's ID
                            const string            name="ArrowCheck", // object name
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
```

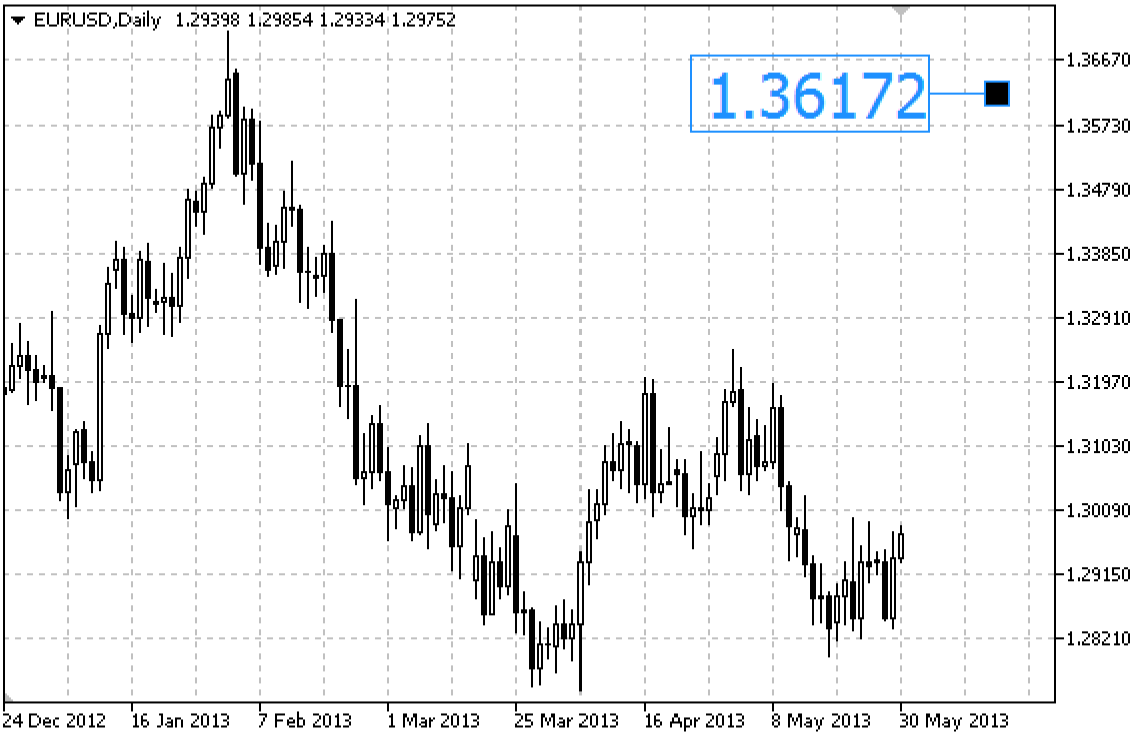
```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Check sign                                                |
//+------------------------------------------------------------------+
bool ArrowCheckDelete(const long   chart_ID=0,        // chart's ID
                      const string name="ArrowCheck") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Check\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
```

```
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Check sign on the chart
   if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the sign
//--- loop counter
   int h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
```

```
       //--- redraw the chart
       ChartRedraw();
       // 0.025 seconds of delay
       Sleep(25);
      }
//--- change anchor point location relative to the sign
   ArrowCheckAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- loop counter
   h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
      {
       //--- use the following value
       if(d<bars-1)
          d+=1;
       //--- move the point
       if(!ArrowCheckMove(0,InpName,date[d],price[p]))
          return;
       //--- check if the script's operation has been forcefully disabled
       if(IsStopped())
          return;
       //--- redraw the chart
       ChartRedraw();
       // 0.025 seconds of delay
       Sleep(25);
      }
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowCheckDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_LEFT_PRICE

Left Price Label


EURUSD,Daily 1.29398 1.29854 1.29334 1.29752
1.36172
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Example

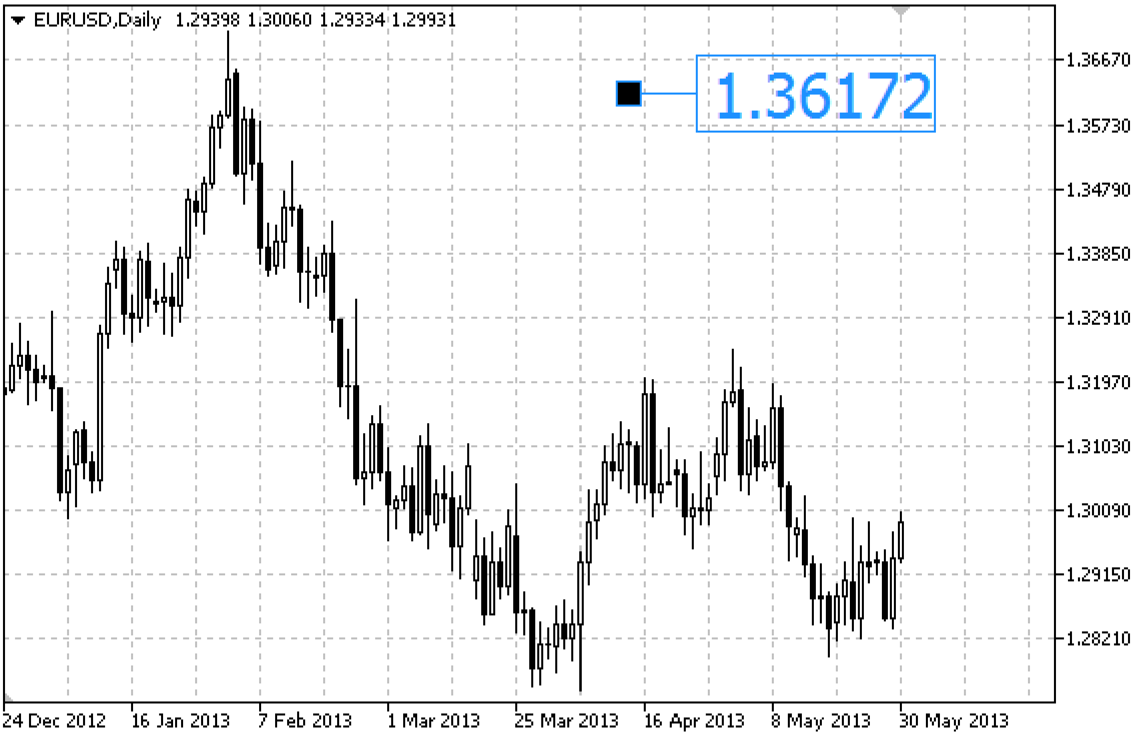
The following script creates and moves left price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the left price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="LeftPrice";    // Price label name
input int               InpDate=100;            // Anchor point date in %
input int               InpPrice=10;            // Anchor point price in %
input color             InpColor=clrRed;        // Price label color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;   // Border line style
input int               InpWidth=2;             // Price label size
input bool              InpBack=false;          // Background label
input bool              InpSelection=true;      // Highlight to move
input bool              InpHidden=true;         // Hidden in the object list
input long              InpZOrder=0;            // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the left price label                                      |
//+------------------------------------------------------------------+
bool ArrowLeftPriceCreate(const long          chart_ID=0,        // chart's ID
                          const string        name="LeftPrice",  // price label nam
                          const int           sub_window=0,      // subwindow index
                          datetime            time=0,            // anchor point ti
                          double              price=0,           // anchor point pr
                          const color         clr=clrRed,        // price label col
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // border line sty
                          const int           width=1,           // price label siz
                          const bool          back=false,        // in the backgrou
                          const bool          selection=true,    // highlight to mo
                          const bool          hidden=true,       // hidden in the o
                          const long          z_order=0)         // priority for mo
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a price label
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create the left price label! Error code = ",GetLastError());
      return(false);
     }
//--- set the label color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the label size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowLeftPriceMove(const long   chart_ID=0,       // chart's ID
                        const string name="LeftPrice", // label name
                        datetime     time=0,           // anchor point time coordinate
                        double       price=0)          // anchor point price coordinat
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the left price label from the chart                       |
//+------------------------------------------------------------------+
bool ArrowLeftPriceDelete(const long   chart_ID=0,       // chart's ID
                          const string name="LeftPrice") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the left price label! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
```

```
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the label
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create the left price label on the chart
   if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point
//--- loop counter
   int v_steps=accuracy*4/5;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowLeftPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label from the chart
   ArrowLeftPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_RIGHT_PRICE

Right Price Label.



Example

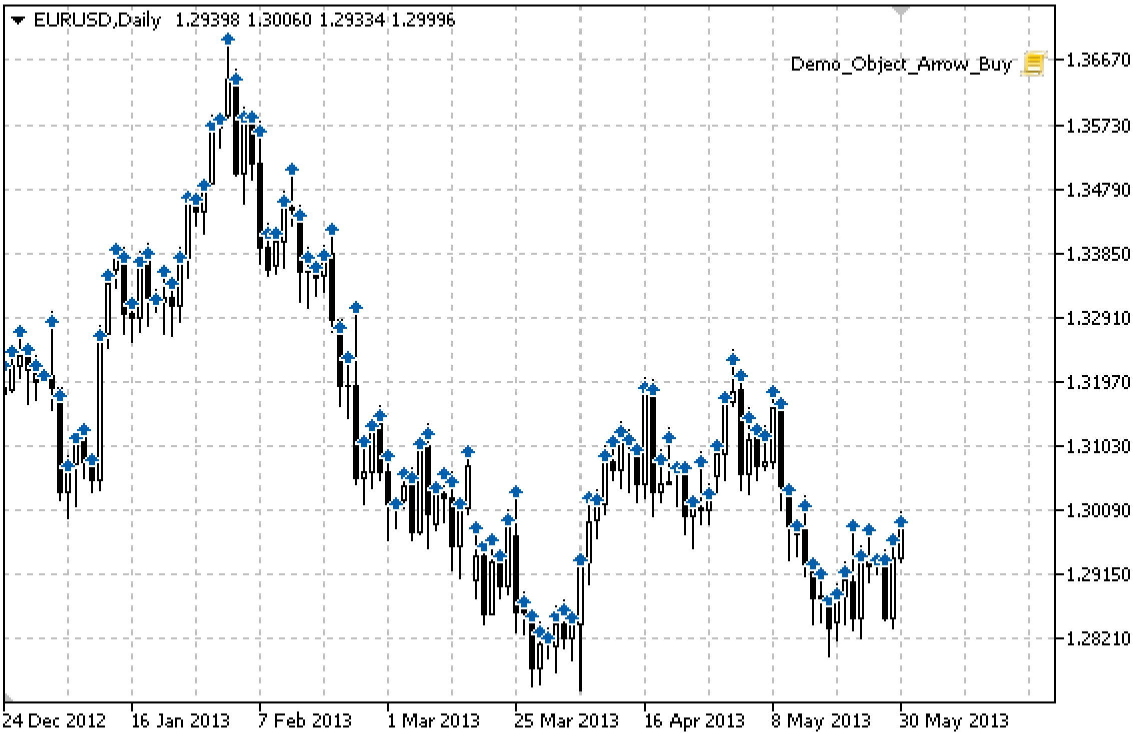
The following script creates and moves right price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the right price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="RightPrice"; // Price label name
input int               InpDate=0;            // Anchor point date in %
input int               InpPrice=90;          // Anchor point price in %
input color             InpColor=clrRed;      // Price label color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // Border line style
input int               InpWidth=2;           // Price label size
input bool              InpBack=false;        // Background label
input bool              InpSelection=true;    // Highlight to move
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the right price label                                     |
//+--------------------------------------------------------------------+
bool ArrowRightPriceCreate(const long            chart_ID=0,        // chart's ID
                           const string          name="RightPrice", // price label na
                           const int             sub_window=0,      // subwindow inde
                           datetime              time=0,            // anchor point t
                           double                price=0,           // anchor point p
                           const color           clr=clrRed,        // price label co
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // border line st
                           const int             width=1,           // price label si
                           const bool            back=false,        // in the backgro
                           const bool            selection=true,    // highlight to m
                           const bool            hidden=true,       // hidden in the
                           const long            z_order=0)         // priority for m
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a price label
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create the right price label! Error code = ",GetLastError());
      return(false);
     }
//--- set the label color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the label size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------+
//| Move the anchor point                                              |
//+--------------------------------------------------------------------+
bool ArrowRightPriceMove(const long   chart_ID=0,        // chart's ID
                         const string name="RightPrice", // label name
                         datetime     time=0,            // anchor point time coordina
                         double       price=0)           // anchor point price coordin
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the right price label from the chart                      |
//+------------------------------------------------------------------+
bool ArrowRightPriceDelete(const long   chart_ID=0,        // chart's ID
                           const string name="RightPrice") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the right price label! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
```

```
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the label
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create the right price label on the chart
   if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point
//--- loop counter
   int v_steps=accuracy*4/5;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowRightPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label from the chart
   ArrowRightPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_BUY

Buy sign.



Example

The following script creates and moves Buy sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Buy\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'3,95,172'; // Color of signs
//+------------------------------------------------------------------+
//| Create Buy sign                                                  |
//+------------------------------------------------------------------+
bool ArrowBuyCreate(const long            chart_ID=0,         // chart's ID
                    const string          name="ArrowBuy",    // sign name
                    const int             sub_window=0,       // subwindow index
                    datetime              time=0,             // anchor point time
                    double                price=0,            // anchor point price
                    const color           clr=C'3,95,172',    // sign color
                    const ENUM_LINE_STYLE style=STYLE_SOLID,  // line style (when highl
                    const int             width=1,            // line size (when highli
                    const bool            back=false,         // in the background
                    const bool            selection=false,    // highlight to move
                    const bool          hidden=true,      // hidden in the object
                    const long          z_order=0)        // priority for mouse cl
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Buy\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowBuyMove(const long   chart_ID=0,      // chart's ID
                  const string name="ArrowBuy", // object name
                  datetime     time=0,          // anchor point time coordinate
                  double       price=0)         // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
```

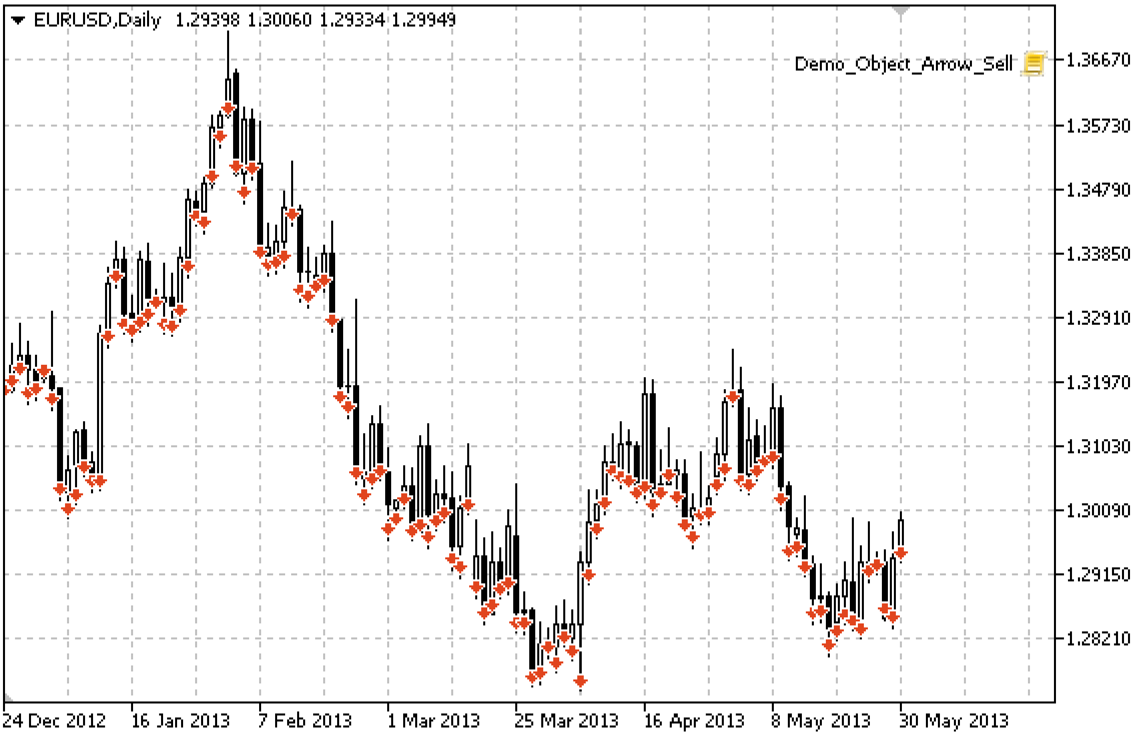
```
      {
       Print(__FUNCTION__,
             ": failed to move the anchor point! Error code = ",GetLastError());
       return(false);
      }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Buy sign                                                  |
//+------------------------------------------------------------------+
bool ArrowBuyDelete(const long   chart_ID=0,      // chart's ID
                    const string name="ArrowBuy") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
      {
       Print(__FUNCTION__,
             ": failed to delete \"Buy\" sign! Error code = ",GetLastError());
       return(false);
      }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

```
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError())
      return;
     }
//--- create Buy signs in Low point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- move Buy signs to High point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
```

```
//--- delete Buy signs
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW_SELL

Sell sign.



Example

The following script creates and moves Sell sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Sell\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'225,68,29'; // Color of signs
//+------------------------------------------------------------------+
//| Create Sell sign                                                 |
//+------------------------------------------------------------------+
bool ArrowSellCreate(const long            chart_ID=0,        // chart's ID
                     const string          name="ArrowSell",  // sign name
                     const int             sub_window=0,      // subwindow index
                     datetime              time=0,            // anchor point time
                     double                price=0,           // anchor point price
                     const color           clr=C'225,68,29',  // sign color
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // line style (when high
                     const int             width=1,           // line size (when highl
                     const bool            back=false,        // in the background
                     const bool            selection=false,   // highlight to move
```

```
                      const bool            hidden=true,       // hidden in the object
                      const long            z_order=0)         // priority for mouse cl
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Sell\" sign! Error code = ",GetLastError());
      return(false);
     }
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowSellMove(const long   chart_ID=0,       // chart's ID
                   const string name="ArrowSell", // object name
                   datetime     time=0,           // anchor point time coordinate
                   double       price=0)          // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
```

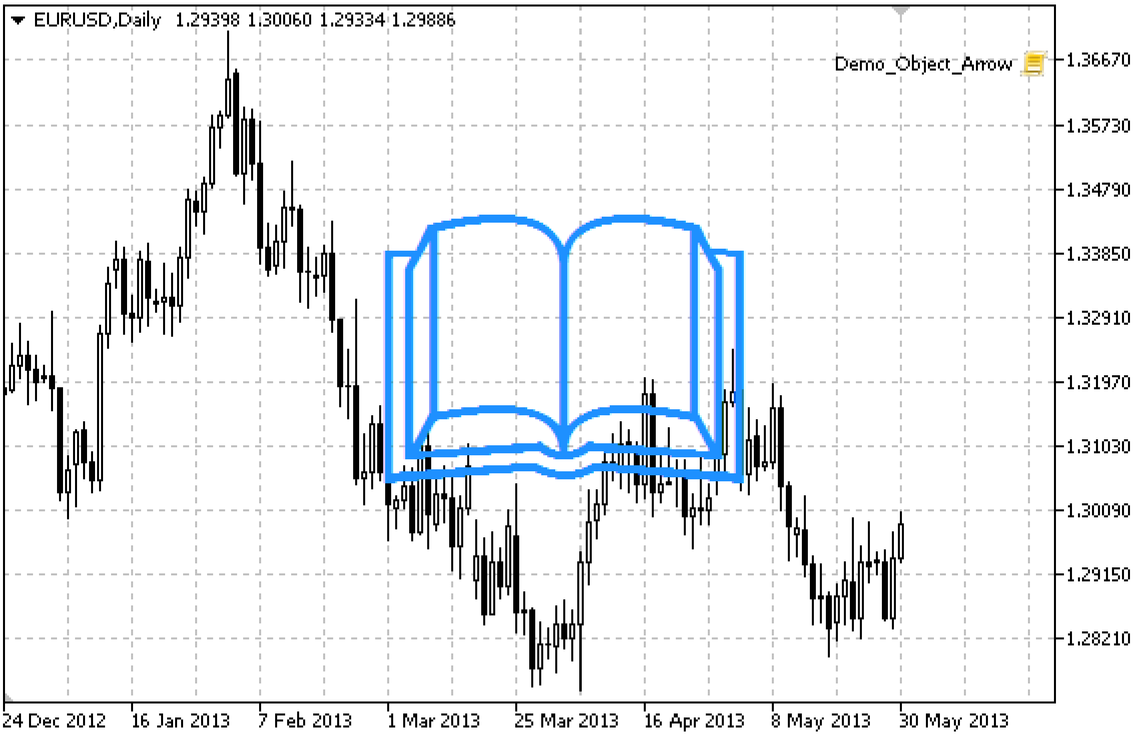
```
      {
       Print(__FUNCTION__,
             ": failed to move the anchor point! Error code = ",GetLastError());
       return(false);
      }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Sell sign                                                 |
//+------------------------------------------------------------------+
bool ArrowSellDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowSell") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
      {
       Print(__FUNCTION__,
             ": failed to delete \"Sell\" sign! Error code = ",GetLastError());
       return(false);
      }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

```
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError());
      return;
     }
//--- create Sell signs in High point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpColor))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- move Sell signs to Low point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
```

```
//--- delete Sell signs
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW

Arrow object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29886
Demo_Object_Arrow
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

Anchor point position relative to the object can be selected from ENUM_ARROW_ANCHOR.

Large arrows (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

The necessary arrow type can be selected by setting one of the Wingdings font's symbol codes.

Example

The following script creates Arrow object on the chart and changes its type. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates a random arrow in the chart window."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="Arrow";        // Arrow name
input int               InpDate=50;             // Anchor point date in %
input int               InpPrice=50;            // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP;   // Anchor type
input color             InpColor=clrDodgerBlue; // Arrow color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;   // Border line style
input int               InpWidth=10;            // Arrow size
input bool              InpBack=false;          // Background arrow
input bool              InpSelection=false;     // Highlight to move
input bool              InpHidden=true;         // Hidden in the object list
input long              InpZOrder=0;            // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the arrow                                                 |
//+------------------------------------------------------------------+
bool ArrowCreate(const long              chart_ID=0,           // chart's ID
                 const string            name="Arrow",         // arrow name
                 const int               sub_window=0,         // subwindow index
                 datetime                time=0,               // anchor point time
                 double                  price=0,              // anchor point price
                 const uchar             arrow_code=252,       // arrow code
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor point positi
                 const color             clr=clrRed,           // arrow color
                 const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border line style
                 const int               width=3,              // arrow size
                 const bool              back=false,           // in the background
                 const bool              selection=true,       // highlight to move
                 const bool              hidden=true,          // hidden in the object
                 const long              z_order=0)            // priority for mouse
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create an arrow
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- set the arrow code
   ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set the arrow color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the arrow's size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the arrow by mouse
```

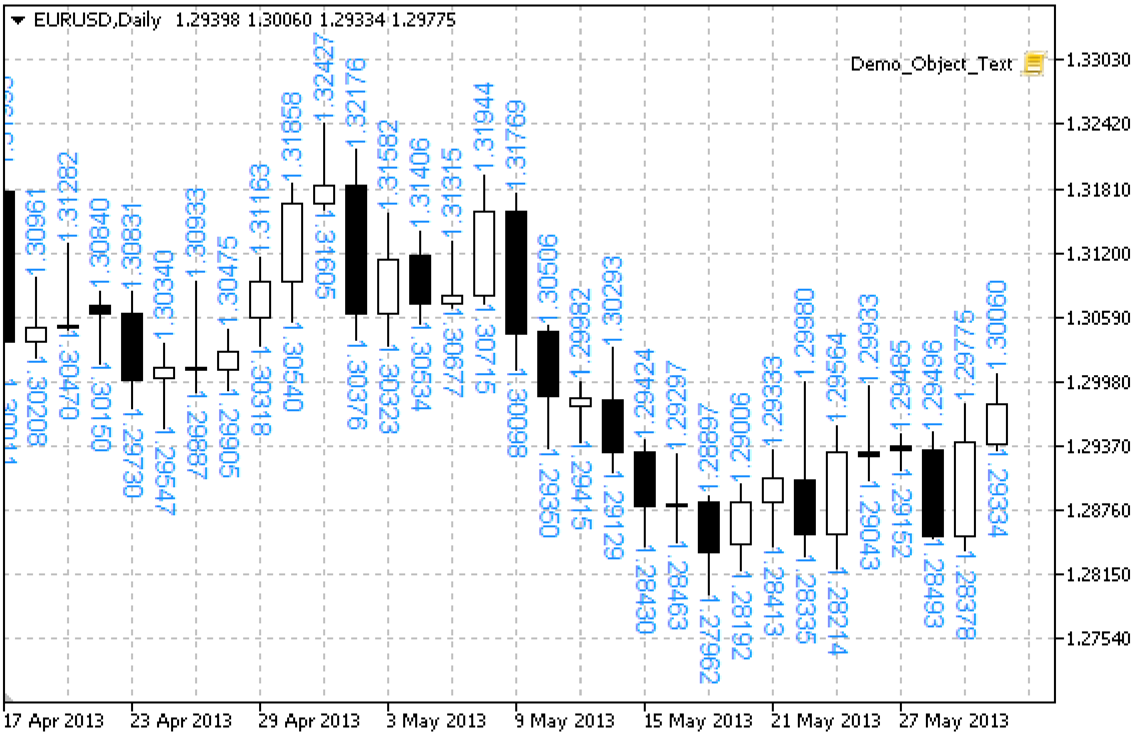
```
//--- when creating a graphical object using ObjectCreate function, the object cannot
//--- highlighted and moved by default. Inside this method, selection parameter
//--- is true by default making it possible to highlight and move the object
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowMove(const long   chart_ID=0,    // chart's ID
               const string name="Arrow", // object name
               datetime     time=0,        // anchor point time coordinate
               double       price=0)       // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the arrow code                                            |
//+------------------------------------------------------------------+
bool ArrowCodeChange(const long   chart_ID=0,    // chart's ID
                     const string name="Arrow", // object name
                     const uchar  code=252)      // arrow code
  {
//--- reset the error value
   ResetLastError();
//--- change the arrow code
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))
     {
      Print(__FUNCTION__,
            ": failed to change the arrow code! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change anchor type                                               |
//+------------------------------------------------------------------+
bool ArrowAnchorChange(const long              chart_ID=0,        // chart's ID
                       const string            name="Arrow",      // object name
                       const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anchor type
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete an arrow                                                  |
//+------------------------------------------------------------------+
bool ArrowDelete(const long   chart_ID=0,    // chart's ID
                 const string name="Arrow") // arrow name
  {
//--- reset the error value
   ResetLastError();
//--- delete an arrow
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
```

```
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the arrow
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create an arrow on the chart
```

```
   if(!ArrowCreate(0,InpName,0,date[d],price[p],32,InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart
   ChartRedraw();
//--- consider all cases of creating arrows in the loop
   for(int i=33;i<256;i++)
     {
      if(!ArrowCodeChange(0,InpName,(uchar)i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // half a second of delay
      Sleep(500);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the arrow from the chart
   ArrowDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TEXT

Text object.



Note

Anchor point position relative to the text can be selected from ENUM_ANCHOR_POINT enumeration. You can also change text slope angle using OBJPROP_ANGLE property.

Example

The following script creates several Text objects on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

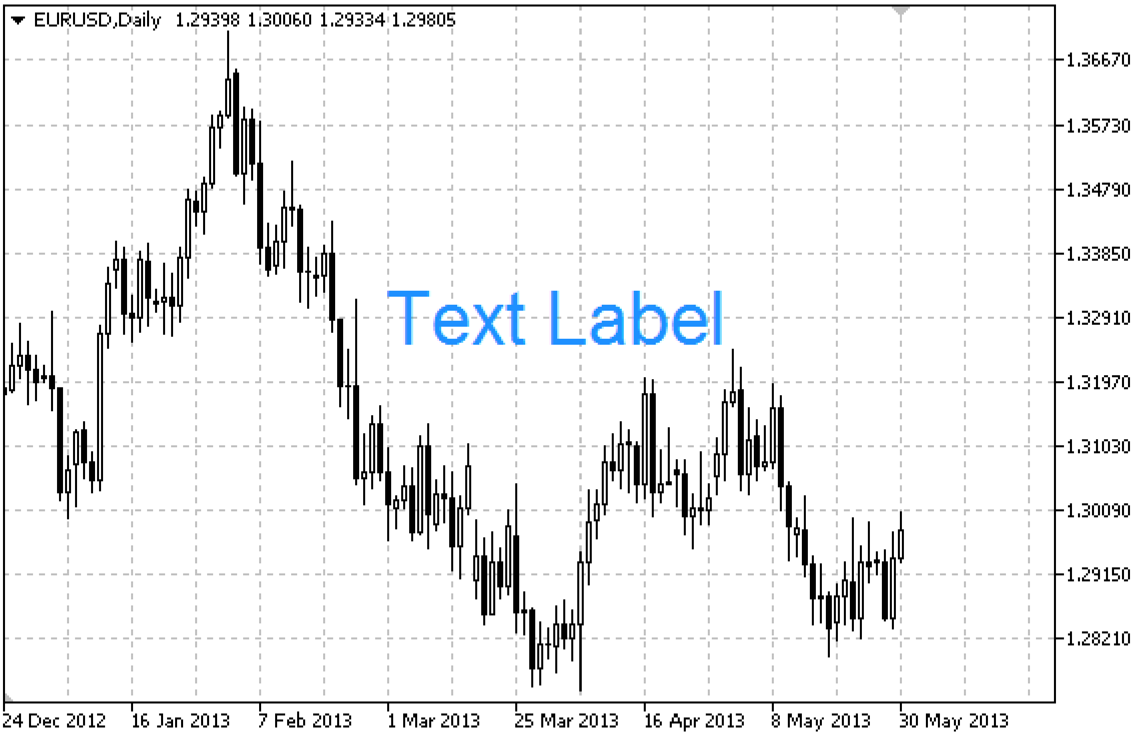
```
//--- description
#property description "Script creates \"Text\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpFont="Arial";          // Font
input int               InpFontSize=10;           // Font size
input color             InpColor=clrRed;          // Color
input double            InpAngle=90.0;            // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_LEFT;    // Anchor type
input bool              InpBack=false;            // Background object
input bool              InpSelection=false;       // Highlight to move
input bool              InpHidden=true;           // Hidden in the object list
input long              InpZOrder=0;              // Priority for mouse click
```

```
//+------------------------------------------------------------------+
//| Creating Text object                                             |
//+------------------------------------------------------------------+
bool TextCreate(const long              chart_ID=0,               // chart's ID
                const string            name="Text",              // object name
                const int               sub_window=0,             // subwindow index
                datetime                time=0,                   // anchor point time
                double                  price=0,                  // anchor point pric
                const string            text="Text",              // the text itself
                const string            font="Arial",             // font
                const int               font_size=10,             // font size
                const color             clr=clrRed,               // color
                const double            angle=0.0,                // text slope
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor type
                const bool              back=false,               // in the backgroun
                const bool              selection=false,          // highlight to move
                const bool              hidden=true,              // hidden in the obj
                const long              z_order=0)                // priority for mous
  {
//--- set anchor point coordinates if they are not set
   ChangeTextEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create Text object
   if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Text\" object! Error code = ",GetLastError());
      return(false);
     }
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the slope angle of the text
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the object by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool TextMove(const long   chart_ID=0,  // chart's ID
              const string name="Text", // object name
              datetime     time=0,      // anchor point time coordinate
              double       price=0)     // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid price
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the object text                                           |
//+------------------------------------------------------------------+
bool TextChange(const long   chart_ID=0,  // chart's ID
                const string name="Text", // object name
                const string text="Text") // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Delete Text object                                               |
//+------------------------------------------------------------------+
bool TextDelete(const long   chart_ID=0,  // chart's ID
                const string name="Text") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Text\" object! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeTextEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLastError());
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetLastError());
      return;
     }
//--- define how often texts are to be displayed
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- define the step
   int step=1;
   switch(scale)
     {
      case 0:
         step=12;
         break;
      case 1:
         step=6;
         break;
      case 2:
         step=4;
         break;
      case 3:
         step=2;
         break;
     }
//--- create texts for High and Low bars' values (with gaps)
   for(int i=0;i<bars;i+=step)
     {
      //--- create the texts
      if(!TextCreate(0,"TextHigh_"+(string)i,0,date[i],high[i],DoubleToString(high[i],
         InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
        {
         return;
        }
      if(!TextCreate(0,"TextLow_"+(string)i,0,date[i],low[i],DoubleToString(low[i],5),
         InpColor,-InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
        {
         return;
        }
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- delete the texts
   for(int i=0;i<bars;i+=step)
     {
      if(!TextDelete(0,"TextHigh_"+(string)i))
         return;
      if(!TextDelete(0,"TextLow_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_LABEL

Label object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29805
Text Label
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

Anchor point position relative to the label can be selected from ENUM_ANCHOR_POINT enumeration. Anchor point coordinates are set in pixels.

You can also select text label anchoring corner from ENUM_BASE_CORNER enumeration.

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Label\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="Label";         // Label name
input int               InpX=150;                // X-axis distance
input int               InpY=150;                // Y-axis distance
input string            InpFont="Arial";         // Font
input int               InpFontSize=14;          // Font size
input color             InpColor=clrRed;         // Color
input double            InpAngle=0.0;            // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Anchor type
input bool              InpBack=false;          // Background object
input bool              InpSelection=true;      // Highlight to move
input bool              InpHidden=true;         // Hidden in the object list
input long              InpZOrder=0;            // Priority for mouse click
//+-------------------------------------------------------------------+
//| Create a text label                                               |
//+-------------------------------------------------------------------+
bool LabelCreate(const long              chart_ID=0,               // chart's ID
                 const string            name="Label",             // label name
                 const int               sub_window=0,             // subwindow index
                 const int               x=0,                      // X coordinate
                 const int               y=0,                      // Y coordinate
                 const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // chart corner for
                 const string            text="Label",             // text
                 const string            font="Arial",             // font
                 const int               font_size=10,             // font size
                 const color             clr=clrRed,               // color
                 const double            angle=0.0,                // text slope
                 const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor type
                 const bool              back=false,               // in the backgrou
                 const bool              selection=false,          // highlight to mov
                 const bool              hidden=true,              // hidden in the ob
                 const long              z_order=0)                // priority for mou
  {
//--- reset the error value
   ResetLastError();
//--- create a text label
   if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create text label! Error code = ",GetLastError());
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the slope angle of the text
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set color
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the text label                                              |
//+------------------------------------------------------------------+
bool LabelMove(const long   chart_ID=0,   // chart's ID
               const string name="Label", // label name
               const int    x=0,          // X coordinate
               const int    y=0)          // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the text label
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",GetLastError()
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change corner of the chart for binding the label                 |
//+------------------------------------------------------------------+
bool LabelChangeCorner(const long             chart_ID=0,               // chart's ID
                       const string           name="Label",             // label name
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // chart corne
  {
//--- reset the error value
   ResetLastError();
//--- change anchor corner
```

```
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the label text                                            |
//+------------------------------------------------------------------+
bool LabelTextChange(const long   chart_ID=0,   // chart's ID
                     const string name="Label", // object name
                     const string text="Text")  // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a text label                                              |
//+------------------------------------------------------------------+
bool LabelDelete(const long   chart_ID=0,   // chart's ID
                 const string name="Label") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a text label! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- store the label's coordinates in the local variables
   int x=InpX;
   int y=InpY;
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- check correctness of the input parameters
   if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- prepare initial text for the label
   string text;
   StringConcatenate(text,"Upper left corner: ",x,",",y);
//--- create a text label on the chart
   if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,InpFontSize,
      InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for half a second
   ChartRedraw();
   Sleep(500);
//--- move the label and change its text simultaneously
//--- number of iterations by axes
   int h_steps=(int)(x_distance/2-InpX);
   int v_steps=(int)(y_distance/2-InpY);
//--- move the label down
   for(int i=0;i<v_steps;i++)
     {
      //--- change the coordinate
      y+=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
```

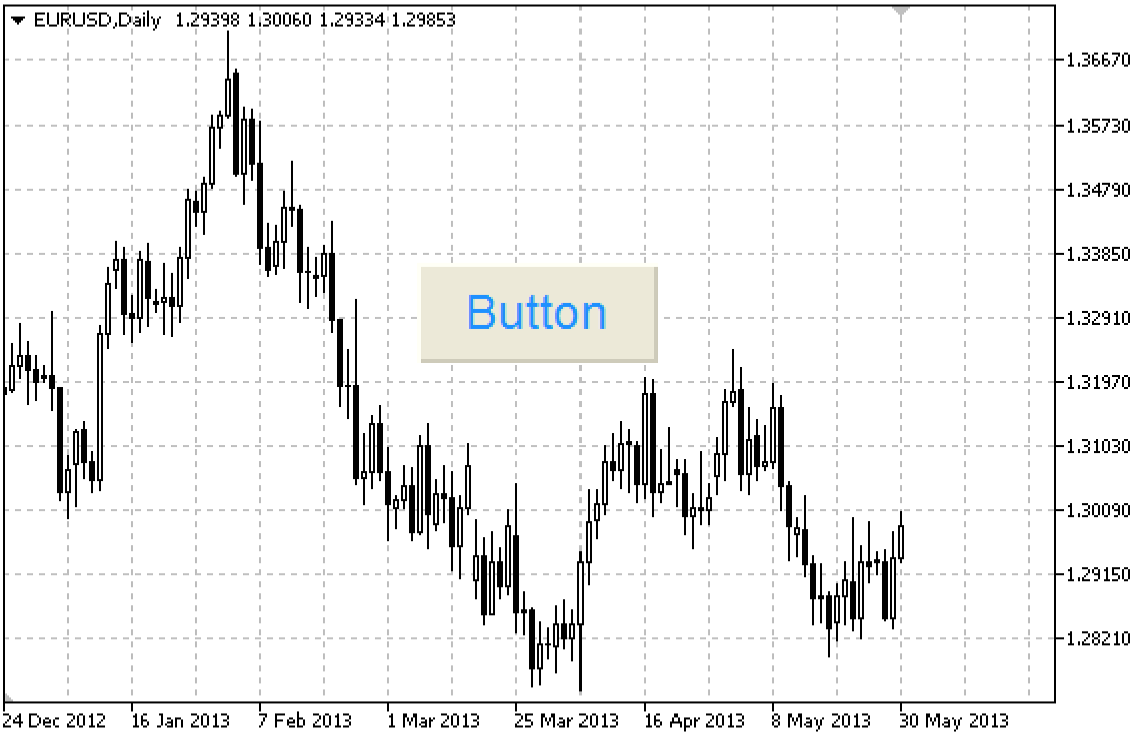
```
//--- half a second of delay
   Sleep(500);
//--- move the label to the right
   for(int i=0;i<h_steps;i++)
     {
      //--- change the coordinate
      x+=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- move the label up
   for(int i=0;i<v_steps;i++)
     {
      //--- change the coordinate
      y-=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- move the label to the left
   for(int i=0;i<h_steps;i++)
     {
      //--- change the coordinate
      x-=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- now, move the point by changing the anchor corner
//--- move to the lower left corner
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
      return;
//--- change the label text
   StringConcatenate(text,"Lower left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the lower right corner
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
      return;
//--- change the label text
   StringConcatenate(text,"Lower right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
```

```
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the upper right corner
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_UPPER))
      return;
//--- change the label text
   StringConcatenate(text,"Upper right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the upper left corner
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_UPPER))
      return;
//--- change the label text
   StringConcatenate(text,"Upper left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- delete the label
   LabelDelete(0,InpName);
//--- redraw the chart and wait for half a second
   ChartRedraw();
   Sleep(500);
//---
  }
//+------------------------------------------------------------------+
//| The function moves the object and changes its text               |
//+------------------------------------------------------------------+
bool MoveAndTextChange(const int x,const int y,string text)
  {
//--- move the label
   if(!LabelMove(0,InpName,x,y))
      return(false);
//--- change the label text
   StringConcatenate(text,text,x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return(false);
//--- check if the script's operation has been forcefully disabled
   if(IsStopped())
      return(false);
//--- redraw the chart
   ChartRedraw();
// 0.01 seconds of delay
```

```
   Sleep(10);
//--- exit the function
   return(true);
  }
```

OBJ_BUTTON

Button object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29853
Button
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

Anchor point coordinates are set in pixels. You can select button anchoring corner from ENUM_BASE_CORNER.

Example

The following script creates and moves Button object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates the button on the chart."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="Button";            // Button name
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input string           InpFont="Arial";             // Font
input int              InpFontSize=14;              // Font size
input color            InpColor=clrBlack;           // Text color
input color            InpBackColor=C'236,233,216'; // Background color
input color            InpBorderColor=clrNONE;      // Border color
input bool             InpState=false;              // Pressed/Released
input bool             InpBack=false;               // Background object
input bool             InpSelection=false;          // Highlight to move
input bool             InpHidden=true;             // Hidden in the object list
input long             InpZOrder=0;                // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the button                                                |
//+------------------------------------------------------------------+
bool ButtonCreate(const long              chart_ID=0,               // chart's ID
                  const string            name="Button",            // button name
                  const int               sub_window=0,             // subwindow inde
                  const int               x=0,                      // X coordinate
                  const int               y=0,                      // Y coordinate
                  const int               width=50,                 // button width
                  const int               height=18,                // button height
                  const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // chart corner fo
                  const string            text="Button",            // text
                  const string            font="Arial",             // font
                  const int               font_size=10,             // font size
                  const color             clr=clrBlack,             // text color
                  const color             back_clr=C'236,233,216',  // background colo
                  const color             border_clr=clrNONE,       // border color
                  const bool              state=false,              // pressed/release
                  const bool              back=false,               // in the backgrou
                  const bool              selection=false,          // highlight to mo
                  const bool              hidden=true,              // hidden in the o
                  const long              z_order=0)                // priority for mo
  {
//--- reset the error value
   ResetLastError();
//--- create the button
   if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create the button! Error code = ",GetLastError());
      return(false);
     }
//--- set button coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set button size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set text color
```

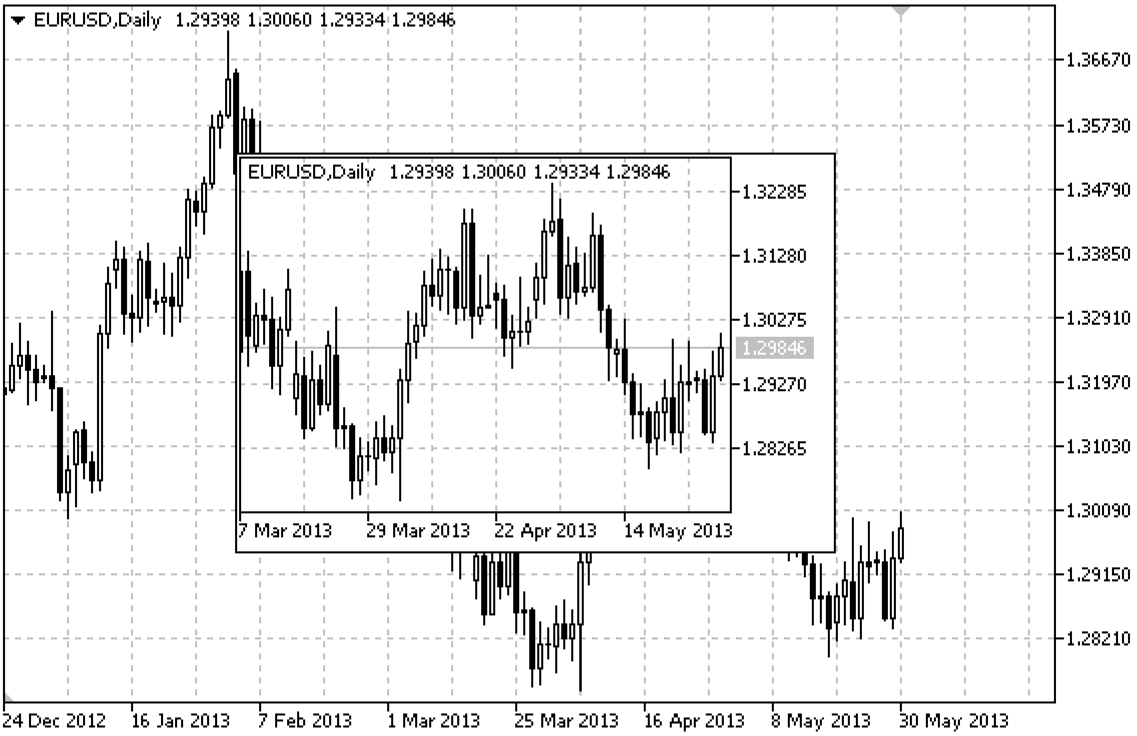
```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- set button state
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- enable (true) or disable (false) the mode of moving the button by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the button                                                  |
//+------------------------------------------------------------------+
bool ButtonMove(const long   chart_ID=0,    // chart's ID
                const string name="Button", // button name
                const int    x=0,           // X coordinate
                const int    y=0)           // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the button
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the button! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the button! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change button size                                               |
//+------------------------------------------------------------------+
bool ButtonChangeSize(const long   chart_ID=0,    // chart's ID
                      const string name="Button", // button name
                      const int    width=50,      // button width
                      const int    height=18)     // button height
  {
//--- reset the error value
   ResetLastError();
//--- change the button size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the button width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the button height! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change corner of the chart for binding the button                |
//+------------------------------------------------------------------+
bool ButtonChangeCorner(const long             chart_ID=0,               // chart's I
                        const string           name="Button",            // button na
                        const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // chart cor
  {
//--- reset the error value
   ResetLastError();
//--- change anchor corner
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change button text                                               |
//+------------------------------------------------------------------+
bool ButtonTextChange(const long   chart_ID=0,    // chart's ID
                      const string name="Button", // button name
                      const string text="Text")   // text
  {
//--- reset the error value
```

```
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the button                                                |
//+------------------------------------------------------------------+
bool ButtonDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Button") // button name
  {
//--- reset the error value
   ResetLastError();
//--- delete the button
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the button! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- define the step for changing the button size
```

```
   int x_step=(int)x_distance/32;
   int y_step=(int)y_distance/32;
//--- set the button coordinates and its size
   int x=(int)x_distance/32;
   int y=(int)y_distance/32;
   int x_size=(int)x_distance*15/16;
   int y_size=(int)y_distance*15/16;
//--- create the button
   if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFont,InpFontSi
      InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,InpHidden,Inp
     {
      return;
     }
//--- redraw the chart
   ChartRedraw();
//--- reduce the button in the loop
   int i=0;
   while(i<13)
     {
      //--- half a second of delay
      Sleep(500);
      //--- switch the button to the pressed state
      ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
      //--- redraw the chart and wait for 0.2 second
      ChartRedraw();
      Sleep(200);
      //--- redefine coordinates and button size
      x+=x_step;
      y+=y_step;
      x_size-=x_step*2;
      y_size-=y_step*2;
      //--- reduce the button
      ButtonMove(0,InpName,x,y);
      ButtonChangeSize(0,InpName,x_size,y_size);
      //--- bring the button back to the released state
      ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
      //--- redraw the chart
      ChartRedraw();
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- increase the loop counter
      i++;
     }
//--- half a second of delay
   Sleep(500);
//--- delete the button
   ButtonDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

OBJ_CHART

Chart object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29846
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013
EURUSD,Daily 1.29398 1.30060 1.29334 1.29846
1.32285
1.31280
1.30275
1.29846
1.29270
1.28265
7 Mar 2013
29 Mar 2013
22 Apr 2013
14 May 2013

Note

" OBJ_CHART" type object is not supported (not displayed) during a visual test.

Anchor point coordinates are set in pixels. You can select anchoring corner from ENUM_BASE_CORNER enumeration.

Symbol, period and scale can be selected for Chart object. Price scale and date display mode can also be enabled/disabled.

Example

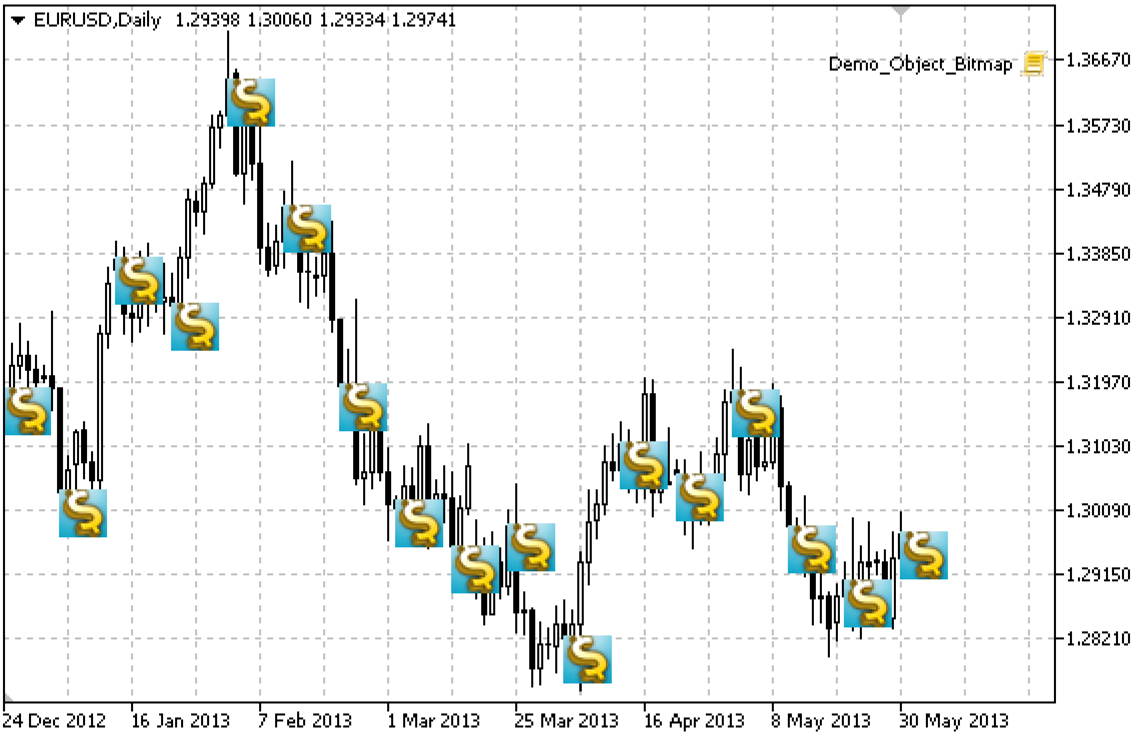
The following script creates and moves Chart object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Chart\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="Chart";              // Object name
input string            InpSymbol="EURUSD";           // Symbol
input ENUM_TIMEFRAMES   InpPeriod=PERIOD_H1;          // Period
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER;  // Anchoring corner
input int               InpScale=2;                   // Scale
input bool             InpDateScale=true;          // Time scale display
input bool             InpPriceScale=true;         // Price scale display
input color            InpColor=clrRed;            // Border color when highlighted
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT;  // Line style when highlighted
input int              InpPointWidth=1;            // Point size to move
input bool             InpBack=false;              // Background object
input bool             InpSelection=true;          // Highlight to move
input bool             InpHidden=true;             // Hidden in the object list
input long             InpZOrder=0;                // Priority for mouse click
//+-------------------------------------------------------------------+
//| Creating Chart object                                             |
//+-------------------------------------------------------------------+
bool ObjectChartCreate(const long              chart_ID=0,               // chart's I
                       const string            name="Chart",             // object na
                       const int               sub_window=0,             // subwindow
                       const string            symbol="EURUSD",          // symbol
                       const ENUM_TIMEFRAMES   period=PERIOD_H1,         // period
                       const int               x=0,                      // X coordin
                       const int               y=0,                      // Y coordin
                       const int               width=300,                // width
                       const int               height=200,               // height
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // anchoring
                       const int               scale=2,                  // scale
                       const bool              date_scale=true,          // time scal
                       const bool              price_scale=true,         // price sca
                       const color             clr=clrRed,               // border co
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // line styl
                       const int               point_width=1,            // move poin
                       const bool              back=false,               // in the ba
                       const bool              selection=false,          // highlight
                       const bool              hidden=true,              // hidden in
                       const long              z_order=0)                // priority
  {
//--- reset the error value
   ResetLastError();
//--- create Chart object
   if(!ObjectCreate(chart_ID,name,OBJ_CHART,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Chart\" object! Error code = ",GetLastError());
      return(false);
     }
//--- set object coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set object size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the chart's corner, relative to which point coordinates are defined
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the symbol
   ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol);
//--- set the period
   ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period);
//--- set the scale
   ObjectSetInteger(chart_ID,name,OBJPROP_CHART_SCALE,scale);
//--- display (true) or hide (false) the time scale
   ObjectSetInteger(chart_ID,name,OBJPROP_DATE_SCALE,date_scale);
//--- display (true) or hide (false) the price scale
   ObjectSetInteger(chart_ID,name,OBJPROP_PRICE_SCALE,price_scale);
//--- set the border color when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Sets the symbol and time frame of the Chart object               |
//+------------------------------------------------------------------+
bool ObjectChartSetSymbolAndPeriod(const long            chart_ID=0,       // chart's
                                   const string          name="Chart",     // object r
                                   const string          symbol="EURUSD",  // symbol
                                   const ENUM_TIMEFRAMES period=PERIOD_H1) // time fra
  {
//--- reset the error value
   ResetLastError();
//--- set Chart object's symbol and time frame
   if(!ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol))
     {
      Print(__FUNCTION__,
            ": failed to set a symbol for \"Chart\" object! Error code = ",GetLastErro
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period))
     {
      Print(__FUNCTION__,
            ": failed to set a period for \"Chart\" object! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Chart object                                                |
//+------------------------------------------------------------------+
bool ObjectChartMove(const long   chart_ID=0,   // chart's ID (not Chart object's one)
                     const string name="Chart", // object name
                     const int    x=0,          // X coordinate
                     const int    y=0)          // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Chart object size                                         |
//+------------------------------------------------------------------+
bool ObjectChartChangeSize(const long   chart_ID=0,   // chart's ID (not Chart object
                           const string name="Chart", // object name
                           const int    width=300,    // width
                           const int    height=200)   // height
  {
//--- reset the error value
   ResetLastError();
//--- change the object size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the width of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the height of \"Chart\" object! Error code = ",GetLast
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Return Chart object's ID                                         |
//+------------------------------------------------------------------+
long ObjectChartGetID(const long   chart_ID=0,   // chart's ID (not Chart object's one
                      const string name="Chart") // object name
  {
//--- prepare the variable to get Chart object's ID
   long id=-1;
//--- reset the error value
   ResetLastError();
//--- get ID
   if(!ObjectGetInteger(chart_ID,name,OBJPROP_CHART_ID,0,id))
     {
      Print(__FUNCTION__,
            ": failed to get \"Chart\" object's ID! Error code = ",GetLastError());
     }
//--- return the result
   return(id);
  }
//+------------------------------------------------------------------+
//| Delete Chart object                                              |
//+------------------------------------------------------------------+
bool ObjectChartDelete(const long   chart_ID=0,   // chart's ID (not Chart object's or
                       const string name="Chart") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the button
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Chart\" object! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of symbols in Market Watch
   int  symbols=SymbolsTotal(true);
//--- check if the symbol with a specified name is present in the symbol list
   bool exist=false;
   for(int i=0;i<symbols;i++)
      if(InpSymbol==SymbolName(i,true))
        {
         exist=true;
         break;
        }
   if(!exist)
     {
      Print("Error! ",InpSymbol," symbol is not present in \"Market Watch\"!");
      return;
     }
//--- check validity of input parameters
   if(InpScale<0 || InpScale>5)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }

//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- set Chart object coordinates and its size
   int x=(int)x_distance/16;
   int y=(int)y_distance/16;
   int x_size=(int)x_distance*7/16;
   int y_size=(int)y_distance*7/16;
//--- create Chart object
   if(!ObjectChartCreate(0,InpName,0,InpSymbol,InpPeriod,x,y,x_size,y_size,InpCorner,I
      InpPriceScale,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- stretch Chart object
   int steps=(int)MathMin(x_distance*7/16,y_distance*7/16);
   for(int i=0;i<steps;i++)
     {
      //--- resize
      x_size+=1;
      y_size+=1;
      if(!ObjectChartChangeSize(0,InpName,x_size,y_size))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.01 seconds
      ChartRedraw();
      Sleep(10);
     }
//--- half a second of delay
   Sleep(500);
//--- change chart's time frame
   if(!ObjectChartSetSymbolAndPeriod(0,InpName,InpSymbol,PERIOD_M1))
      return;
   ChartRedraw();
//--- three seconds of delay
   Sleep(3000);
//--- delete the object
   ObjectChartDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

OBJ_BITMAP

Bitmap object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29741
Demo_Object_Bitmap
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

For Bitmap object, you can select visibility scope of an image.

Example

The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

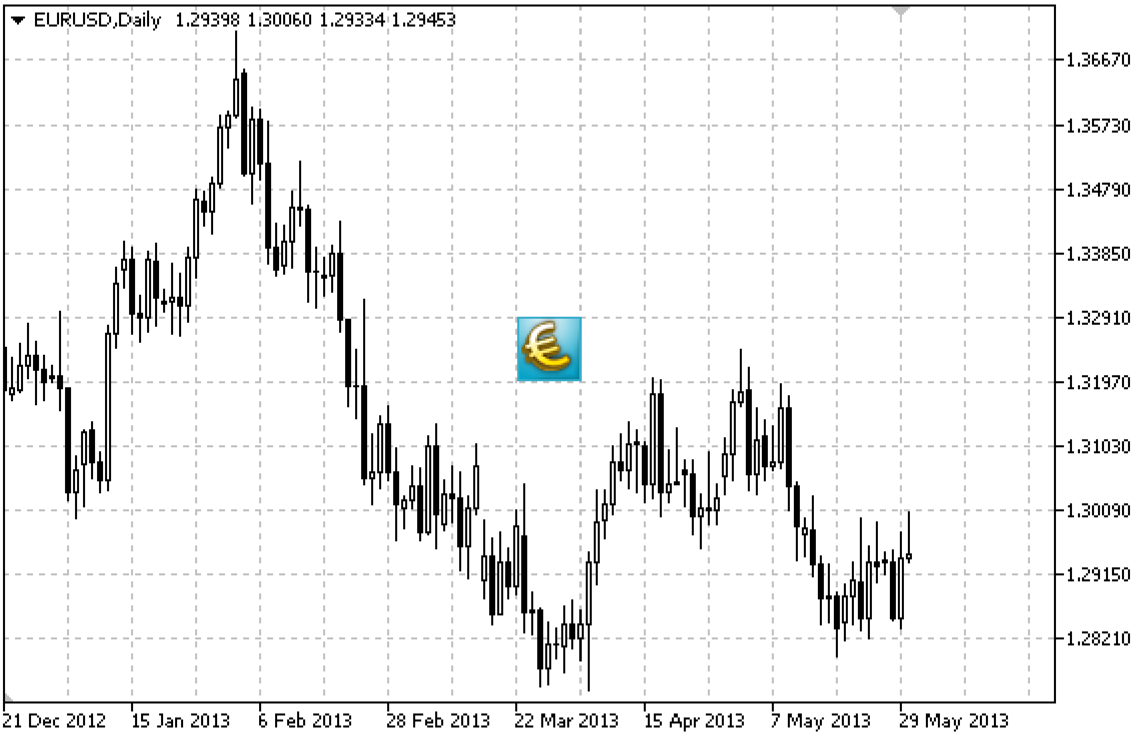
```
//--- description
#property description "Script creates a bitmap in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpFile="\\Images\\dollar.bmp"; // Bitmap file name
input int             InpWidth=24;                    // Visibility scope X coordinat
input int             InpHeight=24;                   // Visibility scope Y coordinat
input int             InpXOffset=4;                   // Visibility scope shift by X
input int             InpYOffset=4;                   // Visibility scope shift by Y
input color           InpColor=clrRed;                // Border color when highlighte
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;           // Line style when highlighted
input int             InpPointWidth=1;                // Point size to move
input bool            InpBack=false;                  // Background object
input bool            InpSelection=false;             // Highlight to move
input bool            InpHidden=true;                // Hidden in the object list
input long            InpZOrder=0;                   // Priority for mouse click
//+------------------------------------------------------------------+
//| Create a bitmap in the chart window                              |
//+------------------------------------------------------------------+
bool BitmapCreate(const long            chart_ID=0,        // chart's ID
                  const string          name="Bitmap",     // bitmap name
                  const int             sub_window=0,      // subwindow index
                  datetime              time=0,            // anchor point time
                  double                price=0,           // anchor point price
                  const string          file="",           // bitmap file name
                  const int             width=10,          // visibility scope X coor
                  const int             height=10,         // visibility scope Y coor
                  const int             x_offset=0,        // visibility scope shift
                  const int             y_offset=0,        // visibility scope shift
                  const color           clr=clrRed,        // border color when highl
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style when highlig
                  const int             point_width=1,     // move point size
                  const bool            back=false,        // in the background
                  const bool            selection=false,   // highlight to move
                  const bool            hidden=true,       // hidden in the object lis
                  const long            z_order=0)         // priority for mouse clic
  {
//--- set anchor point coordinates if they are not set
   ChangeBitmapEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a bitmap
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create a bitmap in the chart window! Error code = ",GetLastE
      return(false);
     }
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility scope
//--- the default part is the upper left area of an image; the values allow
```

```
//--- performing a shift from this area displaying another part of the image
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- set the border color when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set a new image for the bitmap                                   |
//+------------------------------------------------------------------+
bool BitmapSetImage(const long   chart_ID=0,    // chart's ID
                    const string name="Bitmap", // bitmap name
                    const string file="")       // path to the file
  {
//--- reset the error value
   ResetLastError();
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move a bitmap in the chart window                                |
//+------------------------------------------------------------------+
bool BitmapMove(const long   chart_ID=0,    // chart's ID
                const string name="Bitmap", // bitmap name
                datetime     time=0,        // anchor point time
                double       price=0)       // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid price
```

```
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change visibility scope (bitmap) size                            |
//+------------------------------------------------------------------+
bool BitmapChangeSize(const long   chart_ID=0,    // chart's ID
                      const string name="Bitmap", // bitmap name
                      const int    width=0,       // bitmap width
                      const int    height=0)      // bitmap height
  {
//--- reset the error value
   ResetLastError();
//--- change bitmap size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap height! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------+
//| Change coordinate of the upper left corner of the visibility scope |
//+--------------------------------------------------------------------+
bool BitmapMoveVisibleArea(const long   chart_ID=0,    // chart's ID
                           const string name="Bitmap", // bitmap name
                           const int    x_offset=0,    // visibility scope X coordinat
                           const int    y_offset=0)    // visibility scope Y coordinat
  {
//--- reset the error value
   ResetLastError();
//--- change the bitmap's visibility scope coordinates
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Error code = ",
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Error code = ",
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a bitmap                                                  |
//+------------------------------------------------------------------+
bool BitmapDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Bitmap") // bitmap name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a bitmap! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[];  // array for storing dates of visible bars
   double   close[]; // array for storing Close prices
//--- bitmap file name
   string   file="\\Images\\dollar.bmp";
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(close,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Close prices
   if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
     {
      Print("Failed to copy the values of Close prices! Error code = ",GetLastError())
      return;
     }
//--- define how often the images should be displayed
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- define the step
   int step=1;
   switch(scale)
     {
      case 0:
         step=27;
         break;
      case 1:
         step=14;
         break;
      case 2:
         step=7;
         break;
      case 3:
         step=4;
         break;
      case 4:
         step=2;
         break;
     }
//--- create bitmaps for High and Low bars' values (with gaps)
   for(int i=0;i<bars;i+=step)
     {
      //--- create the bitmaps
      if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,InpWidth,InpHe
         InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
        {
         return;
        }
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- delete Sell signs
   for(int i=0;i<bars;i+=step)
     {
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_BITMAP_LABEL

Bitmap Label object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29453
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
21 Dec 2012
15 Jan 2013
6 Feb 2013
28 Feb 2013
22 Mar 2013
15 Apr 2013
7 May 2013
29 May 2013

Note

Anchor point position relative to the label can be selected from ENUM_ANCHOR_POINT enumeration. Anchor point coordinates are set in pixels.

You can also select bitmap anchoring corner from ENUM_BASE_CORNER enumeration.

For bitmap label, you can select visibility scope of an image.

Example

The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Bitmap Label\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="BmpLabel";               // Label name
input string            InpFileOn="\\Images\\dollar.bmp"; // File name for On mode
input string            InpFileOff="\\Images\\euro.bmp";  // File name for Off mode
input bool              InpState=false;                   // Label pressed/released
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER;      // Chart corner for anchori
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER;          // Anchor type
input color             InpColor=clrRed;                  // Border color when highli
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;             // Line style when highlight
input int               InpPointWidth=1;                  // Point size to move
input bool              InpBack=false;                    // Background object
input bool              InpSelection=false;               // Highlight to move
input bool              InpHidden=true;                   // Hidden in the object list
input long              InpZOrder=0;                      // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Bitmap Label object                                       |
//+------------------------------------------------------------------+
bool BitmapLabelCreate(const long              chart_ID=0,               // chart's I
                       const string            name="BmpLabel",          // label name
                       const int               sub_window=0,             // subwindow
                       const int               x=0,                      // X coordina
                       const int               y=0,                      // Y coordina
                       const string            file_on="",               // image in
                       const string            file_off="",              // image in
                       const int               width=0,                  // visibility
                       const int               height=0,                 // visibility
                       const int               x_offset=10,              // visibility
                       const int               y_offset=10,              // visibility
                       const bool              state=false,              // pressed/re
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // chart corn
                       const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anchor typ
                       const color             clr=clrRed,               // border col
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // line style
                       const int               point_width=1,            // move point
                       const bool              back=false,               // in the bac
                       const bool              selection=false,          // highlight
                       const bool              hidden=true,              // hidden in
                       const long              z_order=0)                // priority
  {
//--- reset the error value
   ResetLastError();
//--- create a bitmap label
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Bitmap Label\" object! Error code = ",GetLastError(
      return(false);
     }
//--- set the images for On and Off modes
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,0,file_on))
     {
      Print(__FUNCTION__,
            ": failed to load the image for On mode! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,1,file_off))
     {
      Print(__FUNCTION__,
            ": failed to load the image for Off mode! Error code = ",GetLastError());
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility scope
//--- the default part is the upper left area of an image; the values allow
//--- performing a shift from this area displaying another part of the image
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- define the label's status (pressed or released)
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set the border color when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set a new image for Bitmap label object                          |
//+------------------------------------------------------------------+
bool BitmapLabelSetImage(const long   chart_ID=0,      // chart's ID
                         const string name="BmpLabel", // label name
                         const int    on_off=0,        // modifier (On or Off)
                         const string file="")         // path to the file
  {
//--- reset the error value
   ResetLastError();
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Bitmap Label object                                         |
//+------------------------------------------------------------------+
bool BitmapLabelMove(const long   chart_ID=0,      // chart's ID
                     const string name="BmpLabel", // label name
                     const int    x=0,             // X coordinate
                     const int    y=0)             // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change visibility scope (object) size                            |
//+------------------------------------------------------------------+
bool BitmapLabelChangeSize(const long   chart_ID=0,      // chart's ID
                           const string name="BmpLabel", // label name
                           const int    width=0,         // label width
                           const int    height=0)        // label height
  {
//--- reset the error value
   ResetLastError();
```

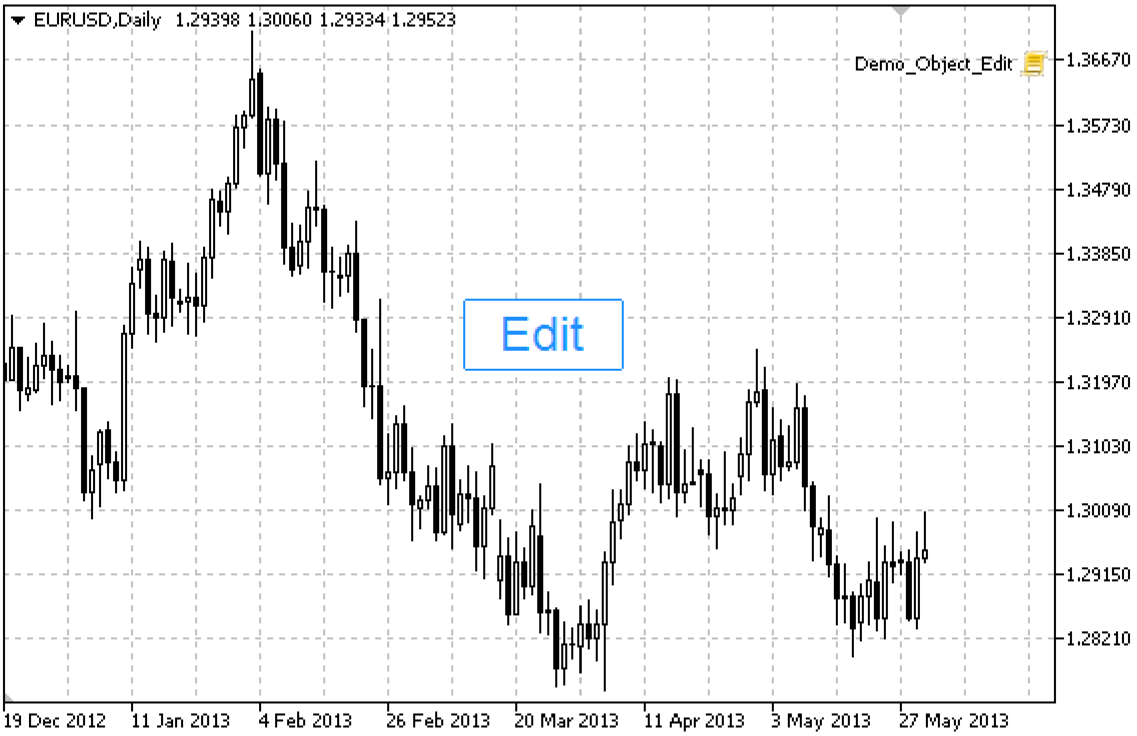
```
//--- change the object size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the object width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the object height! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change coordinate of the upper left corner of the visibility scope |
//+------------------------------------------------------------------+
bool BitmapLabelMoveVisibleArea(const long   chart_ID=0,      // chart's ID
                                const string name="BmpLabel", // label name
                                const int    x_offset=0,      // visibility scope X co
                                const int    y_offset=0)      // visibility scope Y co
  {
//--- reset the error value
   ResetLastError();
//--- change the object's visibility scope coordinates
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Error code = ",G
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Error code = ",G
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete "Bitmap label" object                                     |
//+------------------------------------------------------------------+
bool BitmapLabelDelete(const long   chart_ID=0,      // chart's ID
                       const string name="BmpLabel") // label name
  {
//--- reset the error value
```

```
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Bitmap label\" object! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- define bitmap label coordinates
   int x=(int)x_distance/2;
   int y=(int)y_distance/2;
//--- set label size and visibility scope coordinates
   int width=32;
   int height=32;
   int x_offset=0;
   int y_offset=0;
//--- place bitmap label at the center of the window
   if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height,x_offset,y_
      InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHid
     {
      return;
     }
//--- redraw the chart and wait one second
   ChartRedraw();
   Sleep(1000);
//--- change label's visibility scope size in the loop
   for(int i=0;i<6;i++)
```

```
     {
      //--- change visibility scope size
      width--;
      height--;
      if(!BitmapLabelChangeSize(0,InpName,width,height))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.3 seconds of delay
      Sleep(300);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change label's visibility scope coordinates in the loop
   for(int i=0;i<2;i++)
     {
      //--- change visibility scope coordinates
      x_offset++;
      y_offset++;
      if(!BitmapLabelMoveVisibleArea(0,InpName,x_offset,y_offset))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.3 seconds of delay
      Sleep(300);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label
   BitmapLabelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_EDIT

Edit object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29523
Demo_Object_Edit
Edit
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
19 Dec 2012
11 Jan 2013
4 Feb 2013
26 Feb 2013
20 Mar 2013
11 Apr 2013
3 May 2013
27 May 2013

Note

Anchor point coordinates are set in pixels. You can select Edit anchoring corner from ENUM_BASE_CORNER enumeration.

You can also select one of the text alignment types inside Edit from ENUM_ALIGN_MODE enumeration.

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

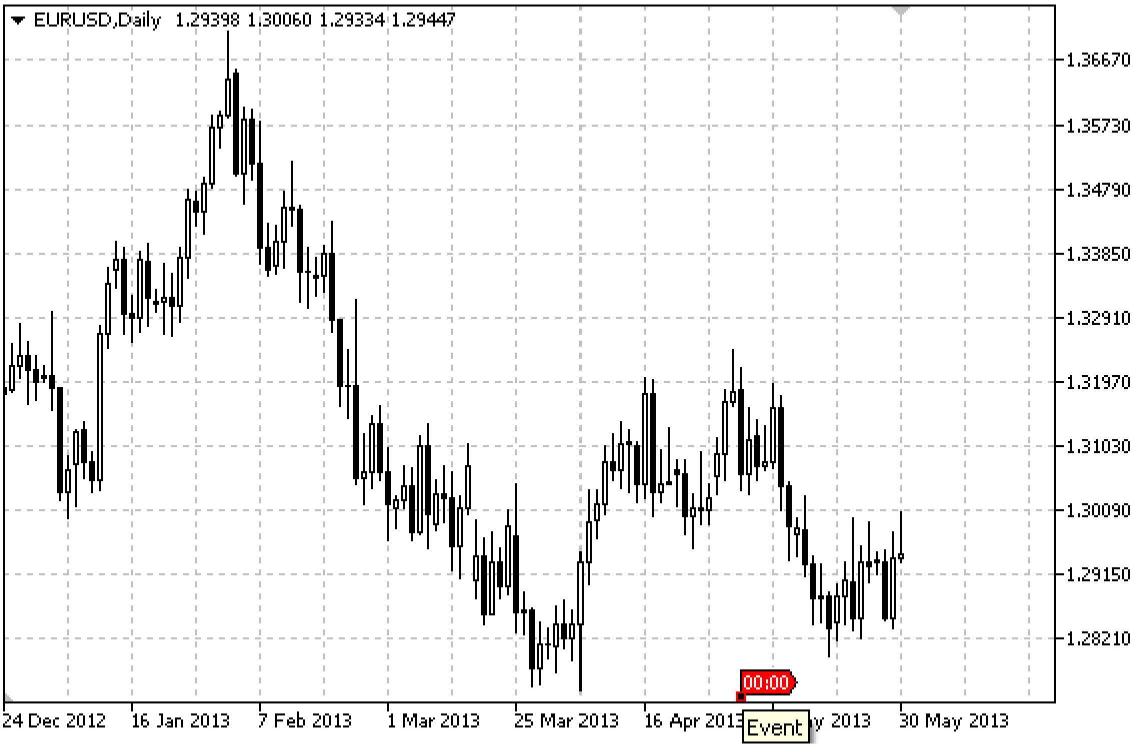
```
//--- description
#property description "Script creates \"Edit\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="Edit";              // Object name
input string           InpText="Text";              // Object text
input string           InpFont="Arial";             // Font
input int              InpFontSize=14;              // Font size
input ENUM_ALIGN_MODE  InpAlign=ALIGN_CENTER;       // Text alignment type
input bool             InpReadOnly=false;           // Permission to edit
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input color            InpColor=clrBlack;           // Text color
input color            InpBackColor=clrWhite;       // Background color
input color            InpBorderColor=clrBlack;     // Border color
input bool             InpBack=false;               // Background object
input bool             InpSelection=false;          // Highlight to move
input bool             InpHidden=true;              // Hidden in the object list
input long             InpZOrder=0;                 // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Edit object                                               |
//+------------------------------------------------------------------+
bool EditCreate(const long             chart_ID=0,               // chart's ID
                const string           name="Edit",              // object name
                const int              sub_window=0,             // subwindow index
                const int              x=0,                      // X coordinate
                const int              y=0,                      // Y coordinate
                const int              width=50,                 // width
                const int              height=18,                // height
                const string           text="Text",              // text
                const string           font="Arial",             // font
                const int              font_size=10,             // font size
                const ENUM_ALIGN_MODE  align=ALIGN_CENTER,       // alignment type
                const bool             read_only=false,          // ability to edit
                const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner for
                const color            clr=clrBlack,             // text color
                const color            back_clr=clrWhite,        // background color
                const color            border_clr=clrNONE,       // border color
                const bool             back=false,               // in the background
                const bool             selection=false,          // highlight to move
                const bool             hidden=true,              // hidden in the obje
                const long             z_order=0)                // priority for mouse
  {
//--- reset the error value
   ResetLastError();
//--- create edit field
   if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Edit\" object! Error code = ",GetLastError());
      return(false);
     }
//--- set object coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set object size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
```

```
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the type of text alignment in the object
   ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- enable (true) or cancel (false) read-only mode
   ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,read_only);
//--- set the chart's corner, relative to which object coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set text color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Edit object                                                 |
//+------------------------------------------------------------------+
bool EditMove(const long   chart_ID=0,  // chart's ID
              const string name="Edit", // object name
              const int    x=0,         // X coordinate
              const int    y=0)         // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Resize Edit object                                               |
//+------------------------------------------------------------------+
bool EditChangeSize(const long   chart_ID=0,  // chart's ID
                    const string name="Edit", // object name
                    const int    width=0,     // width
                    const int    height=0)    // height
  {
//--- reset the error value
   ResetLastError();
//--- change the object size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the object width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the object height! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Edit object's text                                        |
//+------------------------------------------------------------------+
bool EditTextChange(const long   chart_ID=0,  // chart's ID
                    const string name="Edit", // object name
                    const string text="Text") // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Return Edit object text                                          |
//+------------------------------------------------------------------+
bool EditTextGet(string      &text,        // text
                 const long   chart_ID=0,  // chart's ID
                 const string name="Edit") // object name
  {
//--- reset the error value
   ResetLastError();
//--- get object text
   if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
     {
      Print(__FUNCTION__,
            ": failed to get the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Edit object                                               |
//+------------------------------------------------------------------+
bool EditDelete(const long   chart_ID=0,  // chart's ID
                const string name="Edit") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Edit\" object! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- define the step for changing the edit field
   int x_step=(int)x_distance/64;
//--- set edit field coordinates and its size
   int x=(int)x_distance/8;
   int y=(int)y_distance/2;
   int x_size=(int)x_distance/8;
   int y_size=InpFontSize*2;
//--- store the text in the local variable
   string text=InpText;
//--- create edit field
   if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSize,InpAlign,I
      InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,InpHidden,I
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- stretch the edit field
   while(x_size-x<x_distance*5/8)
     {
      //--- increase edit field's width
      x_size+=x_step;
      if(!EditChangeSize(0,InpName,x_size,y_size))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.05 seconds
      ChartRedraw();
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- change the text
   for(int i=0;i<20;i++)
     {
      //--- add "+" at the beginning and at the end
      text="+"+text+"+";
      if(!EditTextChange(0,InpName,text))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
```

```
         return;
      //--- redraw the chart and wait for 0.1 seconds
      ChartRedraw();
      Sleep(100);
     }
//--- half a second of delay
   Sleep(500);
//--- delete edit field
   EditDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

OBJ_EVENT

Event object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29447
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
00:00
Event
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
y 2013
30 May 2013

Note

When hovering mouse over the event, its text appears.

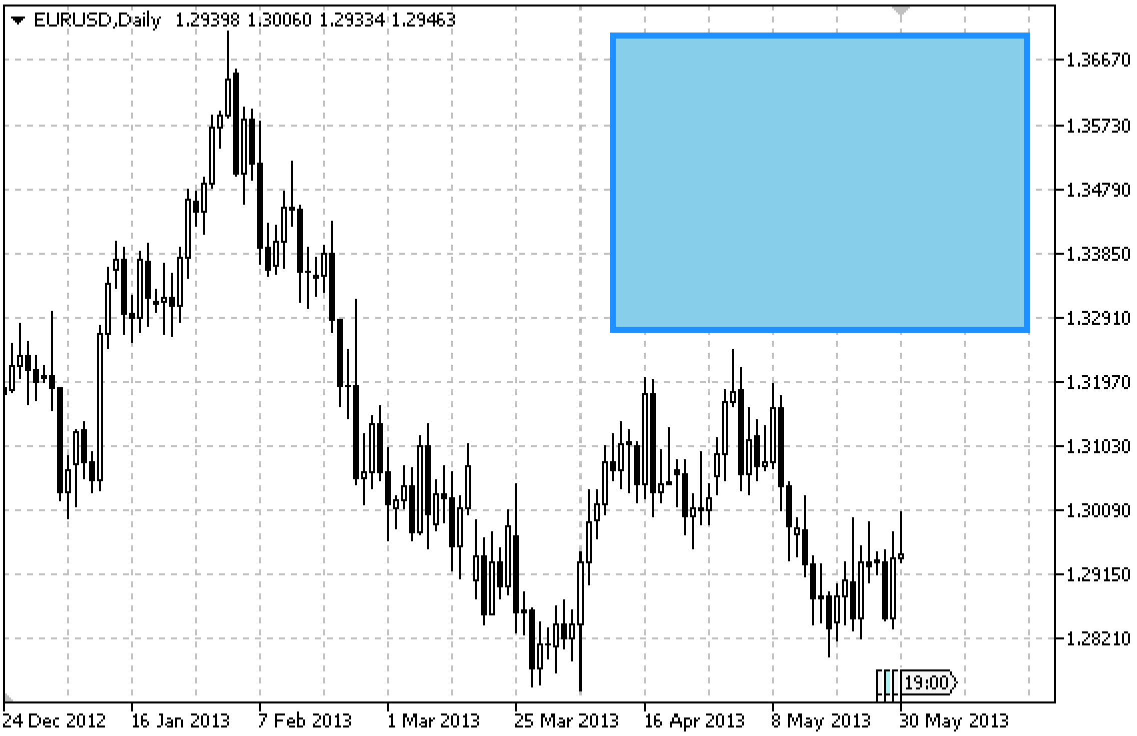
Example

The following script creates and moves Event object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script draws \"Event\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Event";      // Event name
input int             InpDate=25;           // Event date, %
input string          InpText="Text";       // Event text
input color           InpColor=clrRed;      // Event color
input int             InpWidth=1;           // Point size when highlighted
input bool            InpBack=false;        // Background event
input bool            InpSelection=false;   // Highlight to move
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;        // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Event object on the chart                                 |
//+------------------------------------------------------------------+
bool EventCreate(const long            chart_ID=0,      // chart's ID
                 const string          name="Event",    // event name
                 const int             sub_window=0,    // subwindow index
                 const string          text="Text",     // event text
                 datetime              time=0,          // time
                 const color           clr=clrRed,      // color
                 const int             width=1,         // point width when highlighte
                 const bool            back=false,      // in the background
                 const bool            selection=false, // highlight to move
                 const bool            hidden=true,     // hidden in the object list
                 const long            z_order=0)       // priority for mouse click
  {
//--- if time is not set, create the object on the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- create Event object
   if(!ObjectCreate(chart_ID,name,OBJ_EVENT,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Event\" object! Error code = ",GetLastError());
      return(false);
     }
//--- set event text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set anchor point width if the object is highlighted
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving event by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Event object text                                         |
//+------------------------------------------------------------------+
bool EventTextChange(const long   chart_ID=0,   // chart's ID
                     const string name="Event", // event name
                     const string text="Text")  // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Event object                                                |
//+------------------------------------------------------------------+
bool EventMove(const long   chart_ID=0,   // chart's ID
               const string name="Event", // event name
               datetime     time=0)       // time
  {
//--- if time is not set, move event to the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move \"Event\" object! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Event object                                              |
//+------------------------------------------------------------------+
bool EventDelete(const long   chart_ID=0,   // chart's ID
                 const string name="Event") // event name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Event\" object! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing Event object anchor point's coordinates
   datetime date[];
//--- memory allocation
   ArrayResize(date,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- define the points to create an object
   int d=InpDate*(bars-1)/100;
//--- create Event object
   if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the object
//--- loop counter
   int h_steps=bars/2;
//--- move the object
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!EventMove(0,InpName,date[d]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   EventDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE_LABEL

Rectangle Label object.


EURUSD,Daily 1.29398 1.30060 1.29334 1.29463
1.36670
1.35730
1.34790
1.33850
1.32910
1.31970
1.31030
1.30090
1.29150
1.28210
19:00
24 Dec 2012
16 Jan 2013
7 Feb 2013
1 Mar 2013
25 Mar 2013
16 Apr 2013
8 May 2013
30 May 2013

Note

Anchor point coordinates are set in pixels. You can select rectangle label's anchoring corner from ENUM_BASE_CORNER enumeration. Rectangle label's border type can be selected from ENUM_BORDER_TYPE enumeration.

The object is used to create and design the custom graphical interface.

Example

The following script creates and moves Rectangle Label object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
//--- description
#property description "Script creates \"Rectangle Label\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="RectLabel";         // Label name
input color            InpBackColor=clrSkyBlue;     // Background color
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;       // Border type
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for anchoring
input color            InpColor=clrDarkBlue;        // Flat border color (Flat)
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;        // Flat border style (Flat)
input int              InpLineWidth=3;              // Flat border width (Flat)
input bool              InpBack=false;               // Background object
input bool              InpSelection=true;           // Highlight to move
input bool              InpHidden=true;              // Hidden in the object list
input long              InpZOrder=0;                 // Priority for mouse click
//+------------------------------------------------------------------+
//| Create rectangle label                                           |
//+------------------------------------------------------------------+
bool RectLabelCreate(const long             chart_ID=0,               // chart's ID
                     const string           name="RectLabel",         // label name
                     const int              sub_window=0,             // subwindow inc
                     const int              x=0,                      // X coordinate
                     const int              y=0,                      // Y coordinate
                     const int              width=50,                 // width
                     const int              height=18,                // height
                     const color            back_clr=C'236,233,216',  // background cc
                     const ENUM_BORDER_TYPE border=BORDER_SUNKEN,     // border type
                     const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart corner
                     const color            clr=clrRed,               // flat border c
                     const ENUM_LINE_STYLE  style=STYLE_SOLID,        // flat border s
                     const int              line_width=1,             // flat border w
                     const bool             back=false,               // in the backgr
                     const bool             selection=false,          // highlight to
                     const bool             hidden=true,              // hidden in the
                     const long             z_order=0)                // priority for
  {
//--- reset the error value
   ResetLastError();
//--- create a rectangle label
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle label! Error code = ",GetLastError());
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set label size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border type
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- set the chart's corner, relative to which point coordinates are defined
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set flat border color (in Flat mode)
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set flat border line style
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set flat border width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object list
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the chart
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move rectangle label                                             |
//+------------------------------------------------------------------+
bool RectLabelMove(const long   chart_ID=0,       // chart's ID
                   const string name="RectLabel", // label name
                   const int    x=0,              // X coordinate
                   const int    y=0)              // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the rectangle label
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",GetLastError()
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the size of the rectangle label                           |
//+------------------------------------------------------------------+
bool RectLabelChangeSize(const long   chart_ID=0,       // chart's ID
                         const string name="RectLabel", // label name
                         const int    width=50,         // label width
                         const int    height=18)        // label height
  {
```

```
//--- reset the error value
   ResetLastError();
//--- change label size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the label's width! Error code = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the label's height! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change rectangle label border type                               |
//+------------------------------------------------------------------+
bool RectLabelChangeBorderType(const long             chart_ID=0,           // chart'
                               const string           name="RectLabel",     // label
                               const ENUM_BORDER_TYPE border=BORDER_SUNKEN) // border
  {
//--- reset the error value
   ResetLastError();
//--- change border type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
     {
      Print(__FUNCTION__,
            ": failed to change the border type! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the rectangle label                                       |
//+------------------------------------------------------------------+
bool RectLabelDelete(const long   chart_ID=0,       // chart's ID
                     const string name="RectLabel") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a rectangle label! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError());
      return;
     }
//--- define rectangle label coordinates
   int x=(int)x_distance/4;
   int y=(int)y_distance/4;
//--- set label size
   int width=(int)x_distance/4;
   int height=(int)y_distance/4;
//--- create a rectangle label
   if(!RectLabelCreate(0,InpName,0,x,y,width,height,InpBackColor,InpBorder,InpCorner,
      InpColor,InpStyle,InpLineWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait one second
   ChartRedraw();
   Sleep(1000);
//--- change the size of the rectangle label
   int steps=(int)MathMin(x_distance/4,y_distance/4);
   for(int i=0;i<steps;i++)
     {
      //--- resize
      width+=1;
      height+=1;
      if(!RectLabelChangeSize(0,InpName,width,height))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.01 seconds
      ChartRedraw();
      Sleep(10);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change border type
   if(!RectLabelChangeBorderType(0,InpName,BORDER_RAISED))
      return;
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- change border type
   if(!RectLabelChangeBorderType(0,InpName,BORDER_SUNKEN))
      return;
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- delete the label
   RectLabelDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

Object Properties

Graphical objects can have various properties depending on the object type. Values of object properties are set up and received by corresponding functions for working with graphical objects.

All objects used in technical analysis are bound to the time and price coordinates: trendline, channels, Fibonacci tools, etc. But there is a number of auxiliary objects intended to improve the user interface that are bound to the always visible part of a chart (main chart windows or indicator subwindows):

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_CORNER	OBJPROP_ANCHOR	OBJPROP_ANGLE
Text	OBJ_TEXT	—	—	Yes	—	Yes	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	—	Yes	Yes	Yes
Button	OBJ_BUTTON	Yes	Yes	—	Yes	—	—
Bitmap	OBJ_BITMAP	—	Yes (read only)	Yes	—	Yes	—
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	—	Yes	Yes	—
Edit	OBJ_EDIT	Yes	Yes	—	Yes	—	—
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	—	Yes	—	—

The following designations are used in the table:

- **X/Y** – coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** – objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** – anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** – defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** – defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** – defines the object rotation angle counterclockwise.

The functions defining the properties of graphical objects, as well as ObjectCreate() and ObjectMove() operations for creating and moving objects along the chart are actually used for sending commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. Visual changes in the properties of graphical objects are implemented when handling the queue of the chart events.

Thus, do not expect an immediate visual update of graphical objects after calling these functions. Generally, the graphical objects on the chart are updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use ChartRedraw() function to forcefully update the graphical objects.

For functions ObjectSetInteger() and ObjectGetInteger()

ENUM_OBJECT_PROPERTY_INTEGER

Identifier	Description	Property Type
OBJPROP_COLOR	Color	color
OBJPROP_STYLE	Style	ENUM_LINE_STYLE
OBJPROP_WIDTH	Line thickness	int
OBJPROP_BACK	Object in the background	bool
OBJPROP_ZORDER	Priority of a graphical object for receiving events of clicking on a	long

Identifier	Description	Property Type
	chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When objects are placed one atop another, only one	

Identifier	Description	Property Type
	of them with the highest priority will receive the CHARTEVENT_CLICK event.	
OBJPROP_FILL	Fill an object with color (for OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ	bool

Identifier	Description	Property Type
	_STDDEVCHANNEL, OBJ_REGRESSION)	
OBJPROP_HIDDEN	Prohibit showing of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects"	bool

Identifier	Description	Property Type
	. The true value allows to hide an object from the list. By default, true is set to the objects that display calendar events, trading history and to the objects created from MQL	

Identifier	Description	Property Type
	5 programs. To see such graphical objects and access their properties, click on the "All" button in the "List of objects" window.	
OBJPROP_SELECTED	Object is selected	bool
OBJPROP_READONLY	Ability to edit text in	bool

Identifier	Description	Property Type
	the Edit object	
OBJPROP_TYPE	Object type	ENUM_OBJECT r/o
OBJPROP_TIME	Time coordinate	datetime modifier=number of anchor point
OBJPROP_SELECTABLE	Object availability	bool
OBJPROP_CREATETIME	Time of object creation	datetime r/o
OBJPROP_LEVELS	Number of levels	int
OBJPROP_LEVELCOLOR	Color of the line-level	color modifier=level number
OBJPROP_LEVELSTYLE	Style of the line-level	ENUM_LINE_STYLE modifier=level number
OBJPROP_LEVELWIDTH	Thicknes	int modifier=level number

Identifier	Description	Property Type
	s of the line-level	
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	ENUM_ALIGN_MODE
OBJPROP_FONTSIZE	Font size	int
OBJPROP_RAY_LEFT	Ray goes to the left	bool
OBJPROP_RAY_RIGHT	Ray goes to the right	bool
OBJPROP_RAY	A vertical line goes through all the win	bool

Identifier	Description	Property Type
	dows of a chart	
OBJPROP_ELLIPSE	Showing the full ellipse of the Fibonacci Arc object (OBJ_FIBOARC)	bool
OBJPROP_ARROWCODE	Arrow code for the Arrow object	uchar
OBJPROP_TIMEFRAMES	Visibility of an object at timeframes	set of flags flags

Identifier	Description	Property Type
OBJPROP_ANCHOR	Location of the anchor point of a graphical object	ENUM_ARROW_ANCHOR (for OBJ_ARROW), ENUM_ANCHOR_POINT (for OBJ_LABEL, OBJ_BITMAP_LABEL and OBJ_TEXT)
OBJPROP_XDISTANCE	The distance in pixels along the X axis from the binding corner (see note)	int
OBJPROP_YDISTANCE	The distance in pixels along the Y axis	int

Identifier	Description	Property Type
	from the binding corner (see note)	
OBJPROP_DIRECTION	Trend of the Gann object	ENUM_GANN_DIRECTION
OBJPROP_DEGREE	Level of the Elliott Wave Marking	ENUM_ELLIOT_WAVE_DEGREE
OBJPROP_DRAWLINES	Displaying lines for marking the Elliott Wave	bool
OBJPROP_STATE	Button state (pre	bool

Identifier	Description	Property Type
	ssed / depressed)	
OBJPROP_CHART_ID	ID of the "Chart" object (OBJ_CHART). It allows working with the properties of this object like with a normal chart using the functions described in Cha	long r/o

Identifier	Description	Property Type
	rt Operations, but there some exceptions.	
OBJPROP_XSIZE	The object's width along the X axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP	int

Identifier	Description	Property Type
	, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	
OBJPROP_YSIZE	The object's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ	int

Identifier	Description	Property Type
	_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	
OBJPROP_XOFFSET	The X coordinate of the upper left corner of the rectangular visible area in the grap	int

Identifier	Description	Property Type
	hical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	
OBJPROP_YOFFSET	The Y coor	int

Identifier	Description	Property Type
	dinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in	

Identifier	Description	Property Type
	pixels relative to the upper left corner of the original image.	
OBJPROP_PERIOD	Timeframe for the Chart object	ENUM_TIMEFRAMES
OBJPROP_DATE_SCALE	Displaying the time scale for the Chart object	bool
OBJPROP_PRICE_SCALE	Displaying the pric	bool

Identifier	Description	Property Type
	e scale for the Chart object	
OBJPROP_CHART_SCALE	The scale for the Chart object	int value in the range 0-5
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	color
OBJPROP_CORNER	The corner of the chart to	ENUM_BASE_CORNER

Identifier	Description	Property Type
	link a graphical object	
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	ENUM_BORDER_TYPE
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	color

When using chart operations for the "Chart" object (OBJ_CHART), the following limitations are imposed:

- It cannot be closed using ChartClose();
- Symbol/period cannot be changed using the ChartSetSymbolPeriod() function;
- The following properties are ineffective CHART_SCALE, CHART_BRING_TO_TOP, CHART_SHOW_DATE_SCALE and CHART_SHOW_PRICE_SCALE (ENUM_CHART_PROPERTY_INTEGER).

You can set a special mode of image display for OBJ_BITMAP_LABEL and OBJ_BITMAP objects. In this mode, only part of an original image (at which a rectangular visible area is applied) is displayed, while the rest of the image becomes invisible. The size of this area should be set using the properties OBJPROP_XSIZE and OBJPROP_YSIZE. The visible area can be "moved" only within the original image using the properties OBJPROP_XOFFSET and OBJPROP_YOFFSET.

For the fixed-sized objects: OBJ_BUTTON, OBJ_RECTANGLE_LABEL, OBJ_EDIT and OBJ_CHART, properties OBJPROP_XDISTANCE and OBJPROP_YDISTANCE set the position of the top left point of the object relative to the chart corner (OBJPROP_CORNER), from which the X and Y coordinates will be counted in pixels.

For functions ObjectSetDouble() and ObjectGetDouble()

ENUM_OBJECT_PROPERTY_DOUBLE

Identifier	Description	Property Type
OBJPROP_PRICE	Price coordinate	double modifier=number of anchor point
OBJPROP_LEVELVALUE	Level value	double modifier=level number
OBJPROP_SCALE	Scale (properties of Gann objects and Fibonacci Arcs)	double
OBJPROP_ANGLE	Angle.	double

Identifier	Description	Property Type
	For the objects with no angle specified, created from a program, the value is equal to EMPTY_VALUE	
OBJPROP_DEVIATION	Deviation for the Standard Deviation Channel	double

For functions ObjectSetString() and ObjectGetString()

ENUM_OBJECT_PROPERTY_STRING

Identifier	Description	Property Type
OBJPROP_NAME	Object name	string
OBJPROP_TEXT	Description of the object (the text contained in the object)	string
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip generated automatically by the terminal is	string

Identifier	Description	Property Type
	shown. A tooltip can be disabled by setting the "\n" (line feed) value to it	
OBJPROP_LEVELTEXT	Level description	string modifier=level number
OBJPROP_FONT	Font	string
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also Resources	string modifier: 0-state ON, 1-state OFF
OBJPROP_SYMBOL	Symbol for the	string

Identifier	Description	Property Type
	Chart object	

For the OBJ_RECTANGLE_LABEL object ("Rectangle label") one of the three design modes can be set, to which the following values of ENUM_BORDER_TYPE correspond.

ENUM_BORDER_TYPE

Identifier	Description
BORDER_FLAT	Flat form
BORDER_RAISED	Prominent form
BORDER_SUNKEN	Concave form

For the OBJ_EDIT object ("Edit") and for the ChartScreenShot() function, you can specify the horizontal alignment type using the values of the ENUM_ALIGN_MODE enumeration.

ENUM_ALIGN_MODE

Identifier	Description
ALIGN_LEFT	Left alignment
ALIGN_CENTER	Centered (only for the Edit object)
ALIGN_RIGHT	Right alignment

Example:

```
#define  UP          "\x0431"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="my_OBJ_LABEL_object";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object ",label_name," not found. Error code = ",GetLastError());
      //--- create Label object
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- set X coordinate
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- set Y coordinate
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      //--- define text color
      ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
      //--- define text for object Label
      ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
      //--- define font
      ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
      //--- define font size
      ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
      //--- 45 degrees rotation clockwise
      ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);
      //--- disable for mouse selecting
      ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);
      //--- draw it on the chart
      ChartRedraw(0);
     }
  }
```

Methods of Object Binding

Graphical objects Text, Label, Bitmap and Bitmap Label (OBJ_TEXT, OBJ_LABEL, OBJ_BITMAP and OBJ_BITMAP_LABEL) can have one of the 9 different ways of coordinate binding defined by the OBJPROP_ANCHOR property.

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_CORNER	OBJPROP_ANCHOR	OBJPROP_ANGLE
Text	OBJ_TEXT	–	–	Yes	–	**Yes**	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	–	Yes	**Yes**	Yes
Button	OBJ_BUTTON	Yes	Yes	–	Yes	–	–
Bitmap	OBJ_BITMAP	–	Yes (read only)	Yes	–	**Yes**	–
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	–	Yes	**Yes**	–
Edit	OBJ_EDIT	Yes	Yes	–	Yes	–	–
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	–	Yes	–	–

The following designations are used in the table:

- **X/Y** – coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** – objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** – anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** – defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** – defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** – defines the object rotation angle counterclockwise.

The necessary variant can be specified using the function ObjectSetInteger(chart_handle, object_name, OBJPROP_ANCHOR, anchor_point_mode), where anchor_point_mode is one of the values of ENUM_ANCHOR_POINT.

ENUM_ANCHOR_POINT

ID	Description
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner

ID	Description
ANCHOR_LEFT	Anchor point to the left in the center
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner
ANCHOR_LOWER	Anchor point below in the center
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner
ANCHOR_RIGHT	Anchor point to the right in the center
ANCHOR_RIGHT_UPPER	Anchor point at the upper right corner
ANCHOR_UPPER	Anchor point above in the center
ANCHOR_CENTER	Anchor point strictly in the center of the object

The OBJ_BUTTON, OBJ_RECTANGLE_LABEL, OBJ_EDIT and OBJ_CHART objects have a fixed anchor point in the upper left corner (ANCHOR_LEFT_UPPER).

Example:

```
   string text_name="my_OBJ_TEXT_object";
   if(ObjectFind(0,text_name)<0)
     {
      Print("Object ",text_name," not found. Error code = ",GetLastError());
      //--- Get the maximal price of the chart
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- Create object Label
      ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
      //--- Set color of the text
      ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
      //--- Set background color
      ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
      //--- Set text for the Label object
      ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent()));
      //--- Set text font
      ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
      //--- Set font size
      ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
      //--- Bind to the upper right corner
      ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
      //--- Rotate 90 degrees counter-clockwise
      ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
      //--- Forbid the selection of the object by mouse
      ObjectSetInteger(0,text_name,OBJPROP_SELECTABLE,false);
      //--- redraw object
      ChartRedraw(0);
     }
```

Graphical objects Arrow (OBJ_ARROW) have only 2 ways of linking their coordinates. Identifiers are listed in ENUM_ARROW_ANCHOR.

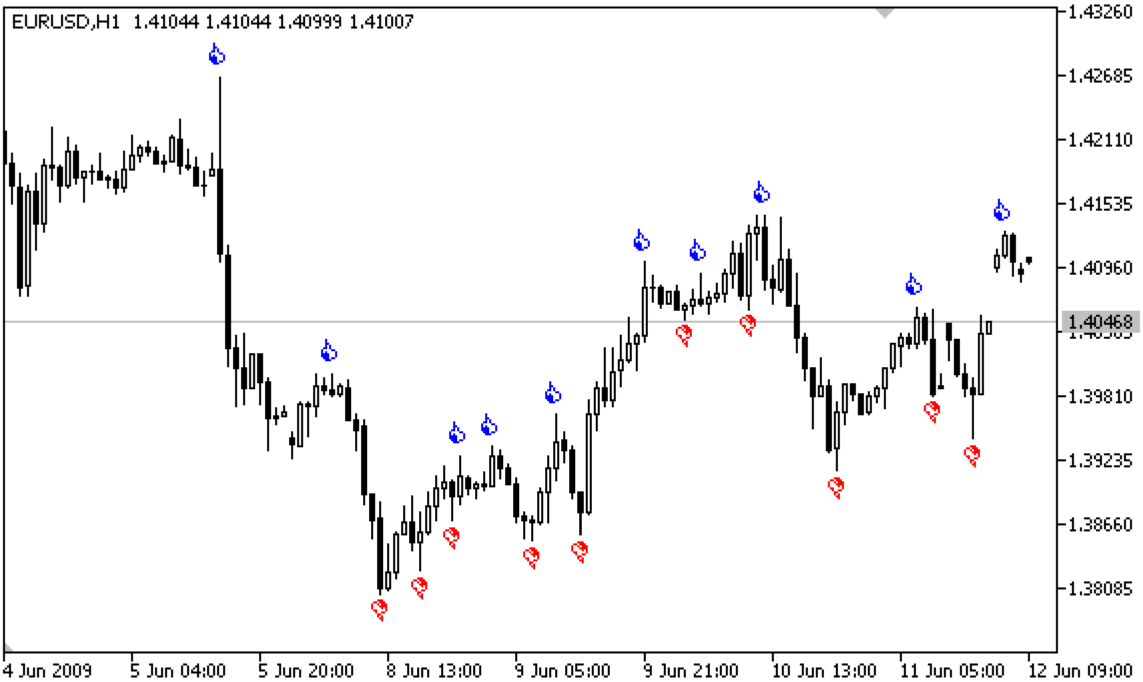
ENUM_ARROW_ANCHOR

ID	Description
ANCHOR_TOP	Anchor on the top side
ANCHOR_BOTTOM	Anchor on the bottom side

Example:

```
void OnStart()
  {
//--- Auxiliary arrays
   double Ups[],Downs[];
   datetime Time[];
//--- Set the arrays as timeseries
   ArraySetAsSeries(Ups,true);
   ArraySetAsSeries(Downs,true);
   ArraySetAsSeries(Time,true);
//--- Create handle of the Indicator Fractals
   int FractalsHandle=iFractals(NULL,0);
   Print("FractalsHandle = ",FractalsHandle);
//--- Set Last error value to Zero
   ResetLastError();
//--- Try to copy the values of the indicator
   int copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Print("Unable to copy the upper fractals. Error = ",GetLastError());
      return;
     }

   ResetLastError();
//--- Try to copy the values of the indicator
   copied=CopyBuffer(FractalsHandle,1,0,1000,Downs);
   if(copied<=0)
     {
      Print("Unable to copy the bottom fractals. Error = ",GetLastError());
      return;
     }

   ResetLastError();
//--- Copy timeseries containing the opening bars of the last 1000 ones
   copied=CopyTime(NULL,0,0,1000,Time);
   if(copied<=0)
     {
      Print("Unable to copy the Opening Time of the last 1000 bars");
      return;
     }
   int upcounter=0,downcounter=0; // count there the number of arrows
   bool created;// receive the result of attempts to create an object
   for(int i=2;i<copied;i++)// Run through the values of the indicator iFractals
     {
      if(Ups[i]!=EMPTY_VALUE)// Found the upper fractal
        {
         if(upcounter<10)// Create no more than 10 "Up" arrows
           {
            //--- Try to create an "Up" object
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_UP,0,Time[i],Ups[i
            if(created)// If set up - let's make tuning for it
              {
               //--- Point anchor is below in order not to cover bar
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_BOTTOM);
               //--- Final touch - painted
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrBlue);
               upcounter++;
              }
           }
        }
      if(Downs[i]!=EMPTY_VALUE)// Found a lower fractal
        {
         if(downcounter<10)// Create no more than 10 arrows "Down"
           {
            //--- Try to create an object "Down"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_DOWN,0,Time[i],Down
            if(created)// If set up - let's make tuning for it
              {
               //--- Point anchor is above in order not to cover bar
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_TOP);
               //--- Final touch - painted
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrRed);
               downcounter++;
              }
           }
        }
     }
  }
```

After the script execution the chart will look like in this figure.


EURUSD,H1 1.41044 1.41044 1.40999 1.41007
1.43260
1.42685
1.42110
1.41535
1.40960
1.40468
1.39810
1.39235
1.38660
1.38085
4 Jun 2009
5 Jun 04:00
5 Jun 20:00
8 Jun 13:00
9 Jun 05:00
9 Jun 21:00
10 Jun 13:00
11 Jun 05:00
12 Jun 09:00

The Chart Corner to Which an Object Is Attached

There is a number of graphical objects for which you can set a chart corner, relative to which the coordinates are specified in pixels. These are the following types of objects (in brackets object type identifiers are specified):

- Label (OBJ_LABEL);
- Button (OBJ_BUTTON);
- Bitmap Label (OBJ_BITMAP_LABEL);
- Edit (OBJ_EDIT).
- Rectangle Label (OBJ_RECTANGLE_LABEL);

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_CORNER	OBJPROP_ANCHOR	OBJPROP_ANGLE
Text	OBJ_TEXT	–	–	Yes	–	Yes	Yes
Label	OBJ_LABEL	Yes	Yes (read only)	–	**Yes**	Yes	Yes
Button	OBJ_BUTTON	Yes	Yes	–	**Yes**	–	–
Bitmap	OBJ_BITMAP	–	Yes (read only)	Yes	–	Yes	–
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)	–	**Yes**	Yes	–
Edit	OBJ_EDIT	Yes	Yes	–	**Yes**	–	–
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes	–	**Yes**	–	–

The following designations are used in the table:

- **X/Y** – coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** – objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** – anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** – defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** – defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** – defines the object rotation angle counterclockwise.

In order to specify the chart corner, from which X and Y coordinates will be measured in pixels, use ObjectSetInteger(chartID, name, OBJPROP_CORNER, **chart_corner**), where:

- chartID - chart identifier;
- name - name of a graphical object;
- OBJPROP_CORNER - property ID to specify the corner for binding;
- **chart_corner** - the desired chart corner, can be one of the values of the ENUM_BASE_CORNER enumeration.

ENUM_BASE_CORNER

ID	Description
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner of the chart

Example:

```
void CreateLabel(long   chart_id,
                 string name,
                 int    chart_corner,
                 int    anchor_point,
                 string text_label,
                 int    x_ord,
                 int    y_ord)
  {
//---
   if(ObjectCreate(chart_id,name,OBJ_LABEL,0,0,0))
     {
      ObjectSetInteger(chart_id,name,OBJPROP_CORNER,chart_corner);
      ObjectSetInteger(chart_id,name,OBJPROP_ANCHOR,anchor_point);
      ObjectSetInteger(chart_id,name,OBJPROP_XDISTANCE,x_ord);
      ObjectSetInteger(chart_id,name,OBJPROP_YDISTANCE,y_ord);
      ObjectSetString(chart_id,name,OBJPROP_TEXT,text_label);
     }
   else
      Print("Failed to create the object OBJ_LABEL ",name,", Error code = ", GetLastE
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int height=(int)ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=(int)ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   string arrows[4]={"LEFT_UPPER","RIGHT_UPPER","RIGHT_LOWER","LEFT_LOWER"};
   CreateLabel(0,arrows[0],CORNER_LEFT_UPPER,ANCHOR_LEFT_UPPER,arrows[0],50,50);
   CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,ANCHOR_RIGHT_UPPER,arrows[1],50,50);
   CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,ANCHOR_RIGHT_LOWER,arrows[2],50,50);
   CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,ANCHOR_LEFT_LOWER,arrows[3],50,50);
  }
```

Visibility of Objects

The combination of object visibility flags determines chart timeframes, where the object is visible. To set/get the value of the OBJPROP_TIMEFRAMES property, you can use functions ObjectSetInteger()/ObjectGetInteger().

ID	Value	Description
OBJ_NO_PERIODS	0	The object is not drawn in all timeframes
OBJ_PERIOD_M1	0x00000001	The object is drawn in 1-minute chart
OBJ_PERIOD_M2	0x00000002	The object is drawn in 2-minute chart
OBJ_PERIOD_M3	0x00000004	The object is drawn in 3-minute chart
OBJ_PERIOD_M4	0x00000008	The object is drawn in 4-minute chart
OBJ_PERIOD_M5	0x00000010	The object is drawn in 5-minute chart
OBJ_PERIOD_M6	0x00000020	The object is drawn in 6-minute chart
OBJ_PERIOD_M10	0x00000040	The object is drawn in 10-minute chart
OBJ_PERIOD_M12	0x00000080	The object is drawn in 12-minute chart
OBJ_PERIOD_M15	0x00000100	The object is drawn in 15-minute chart
OBJ_PERIOD_M20	0x00000200	The object is drawn in 20-minute chart
OBJ_PERIOD_M30	0x00000400	The object is drawn in 30-minute chart
OBJ_PERIOD_H1	0x00000800	The object is drawn in 1-hour chart
OBJ_PERIOD_H2	0x00001000	The object is drawn in 2-hour chart
OBJ_PERIOD_H3	0x00002000	The object is drawn in 3-hour chart
OBJ_PERIOD_H4	0x00004000	The object is drawn in 4-hour chart

ID	Value	Description
OBJ_PERIOD_H6	0x00008000	The object is drawn in 6-hour chart
OBJ_PERIOD_H8	0x00010000	The object is drawn in 8-hour chart
OBJ_PERIOD_H12	0x00020000	The object is drawn in 12-hour chart
OBJ_PERIOD_D1	0x00040000	The object is drawn in day charts
OBJ_PERIOD_W1	0x00080000	The object is drawn in week charts
OBJ_PERIOD_MN1	0x00100000	The object is drawn in month charts
OBJ_ALL_PERIODS	0x001fffff	The object is drawn in all timeframes

Visibility flags can be combined using the symbol "|", for example, the combination of flags OBJ_PERIOD_M10|OBJ_PERIOD_H4 means that the object will be visible on the 10-minute and 4-hour timeframes.

Example:

```
void OnStart()
  {
//---
   string highlevel="PreviousDayHigh";
   string lowlevel="PreviousDayLow";
   double prevHigh;           // The previous day High
   double prevLow;            // The previous day Low
   double highs[],lows[];     // Arrays for High and Low

//--- Reset the last error
   ResetLastError();
//--- Get the last 2 High values on the daily timeframe
   int highsgot=CopyHigh(Symbol(),PERIOD_D1,0,2,highs);
   if(highsgot>0) // If copying was successful
     {
      Print("High prices for the last 2 days were obtained successfully");
      prevHigh=highs[0]; // The previous day High
      Print("prevHigh = ",prevHigh);
      if(ObjectFind(0,highlevel)<0) // Object with the name highlevel not found
        {
         ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // Create the Horizontal Line obje
        }
      //--- Set value for the price level for the line highlevel
      ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
      //--- Set the visibility only PERIOD_M10 and PERIOD_H4
      ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else
     {
      Print("Could not get High prices over the past 2 days, Error = ",GetLastError())
     }

//--- Reset the last error
   ResetLastError();
//--- Get the 2 days values Low on the daily timeframe
   int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2,lows);
   if(lowsgot>0) // If copying was successful
     {
      Print("Low prices for the last 2 days were obtained successfully");
      prevLow=lows[0]; // The previous day Low
      Print("prevLow = ",prevLow);
      if(ObjectFind(0,lowlevel)<0) // Object with the name lowlevel not found
        {
         ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // Create the Horizontal Line objec
        }
      //--- Set value for the price level for the line lowlevel
      ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
      //--- Set the visibility only PERIOD_M10 and PERIOD_H4
      ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else Print("Could not get Low prices for the last 2 days, Error = ",GetLastError())

   ChartRedraw(0); // redraw the chart forcibly
  }
```

See also

PeriodSeconds, Period, Chart timeframes, Date and Time

Levels of Elliott Wave

Elliott Waves are represented by two graphical objects of types OBJ_ELLIOTWAVE5 and OBJ_ELLIOTWAVE3. To set the wave size (method of wave labeling), the OBJPROP_DEGREE property is used, to which one of values of the ENUM_ELLIOT_WAVE_DEGREE enumeration can be assigned.

ENUM_ELLIOT_WAVE_DEGREE

ID	Description
ELLIOTT_GRAND_SUPERCYCLE	Grand Supercycle
ELLIOTT_SUPERCYCLE	Supercycle
ELLIOTT_CYCLE	Cycle
ELLIOTT_PRIMARY	Primary
ELLIOTT_INTERMEDIATE	Intermediate
ELLIOTT_MINOR	Minor
ELLIOTT_MINUTE	Minute
ELLIOTT_MINUETTE	Minuette
ELLIOTT_SUBMINUETTE	Subminuette

Example:

```
   for(int i=0;i<ObjectsTotal(0);i++)
     {
      string currobj=ObjectName(0,i);
      if((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE3) ||
         ((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE5)))
        {
         //--- set the marking level in INTERMEDIATE
         ObjectSetInteger(0,currobj,OBJPROP_DEGREE,ELLIOTT_INTERMEDIATE);
         //--- show lines between tops of waves
         ObjectSetInteger(0,currobj,OBJPROP_DRAWLINES,true);
         //--- set line color
         ObjectSetInteger(0,currobj,OBJPROP_COLOR,clrBlue);
         //--- set line width
         ObjectSetInteger(0,currobj,OBJPROP_WIDTH,5);
         //--- set description
         ObjectSetString(0,currobj,OBJPROP_TEXT,"test script");
        }
     }
```

Gann Objects

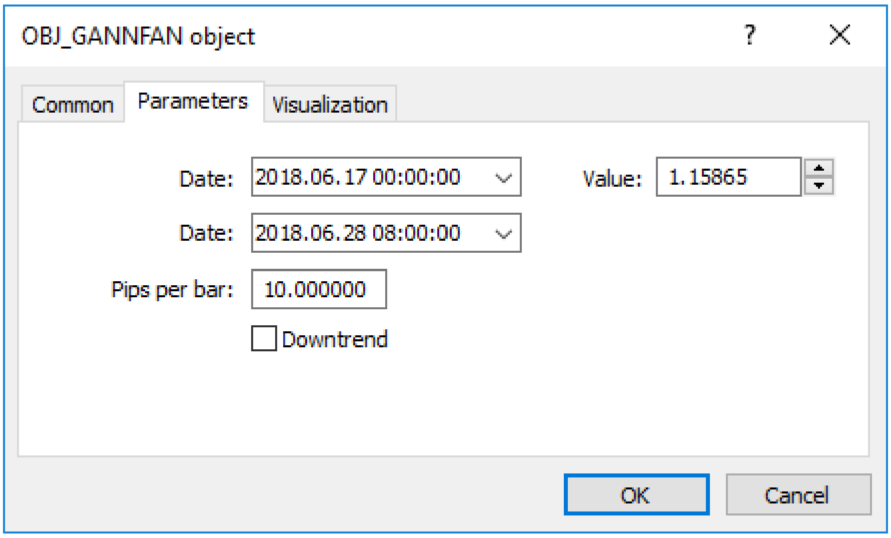
For Gann Fan (OBJ_GANNFAN) and Gann Grid (OBJ_GANNGRID) objects you can specify two values of the ENUM_GANN_DIRECTION enumeration that sets the trend direction.

ENUM_GANN_DIRECTION

ID	Description
GANN_UP_TREND	Line corresponding to the uptrend line
GANN_DOWN_TREND	Line corresponding to the downward trend

To set the scale of the main line as 1x1, use function ObjectSetDouble(chart_handle, gann_object_name, OBJPROP_SCALE, scale), where:

- chart_handle - chart window where the object is located;
- gann_object_name - object name;
- OBJPROP_SCALE - identifier of the "Scale" property;
- scale - required scale in units of Pips/Bar.



Date: 2018.06.17 00:00:00

Value: 1.15865

Date: 2018.06.28 08:00:00

Pips per bar: 10.000000

☐ Downtrend

Example of creating Gann Fan:

```
void OnStart()
  {
//---
   string my_gann="OBJ_GANNFAN object";
   if(ObjectFind(0,my_gann)<0)// Object not found
     {
      //--- Inform about the failure
      Print("Object ",my_gann," not found. Error code = ",GetLastError());
      //--- Get the maximal price of the chart
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- Get the minimal price of the chart
      double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
      //--- How many bars are shown in the chart?
      int bars_on_chart=ChartGetInteger(0,CHART_VISIBLE_BARS);
      //--- Create an array, to write the opening time of each bar to
      datetime Time[];
      //--- Arrange access to the array as that of timeseries
      ArraySetAsSeries(Time,true);
      //--- Now copy data of bars visible in the chart into this array
      int times=CopyTime(NULL,0,0,bars_on_chart,Time);
      if(times<=0)
        {
         Print("Could not copy the array with the open time!");
         return;
        }
      //--- Preliminary preparations completed

      //--- Index of the central bar in the chart
      int center_bar=bars_on_chart/2;
      //--- Chart equator - between the maximum and minimum
      double mean=(chart_max_price+chart_min_price)/2.0;
      //--- Set the coordinates of the first anchor point to the center
      ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Time[center_bar],mean,
                   //--- Second anchor point to the right
                   Time[center_bar/2],(mean+chart_min_price)/2.0);
      Print("Time[center_bar] = "+(string)Time[center_bar]+"  Time[center_bar/2] = "+
      //Print("Time[center_bar]/="+Time[center_bar]+"  Time[center_bar/2]="+Time[cente
      //--- Set the scale in units of Pips / Bar
      ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
      //--- Set the line trend
      ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
      //--- Set the line width
      ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
      //--- Define the line style
      ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
      //--- Set the line color
      ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
      //--- Allow the user to select an object
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
      //--- Select it yourself
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
      //--- Draw it on the chart
      ChartRedraw(0);
     }
  }
```

Web Colors

The following color constants are defined for the color type:

clrBlack	clrDarkGreen	clrDarkSlateGray	clrOlive	clrGreen	clrTeal	clrNavy	clrPurple
clrMaroon	clrIndigo	clrMidnightBlue	clrDarkBlue	clrDarkOliveGreen	clrSaddleBrown	clrForestGreen	clrOliveDrab
clrSeaGreen	clrDarkGoldenrod	clrDarkSlateBlue	clrSienna	clrMediumBlue	clrBrown	clrDarkTurquoise	clrDimGray
clrLightSeaGreen	clrDarkViolet	clrFireBrick	clrMediumVioletRed	clrMediumSeaGreen	clrChocolate	clrCrimson	clrSteelBlue
clrGoldenrod	clrMediumSpringGreen	clrLawnGreen	clrCadetBlue	clrDarkOrchid	clrYellowGreen	clrLimeGreen	clrOrangeRed
clrDarkOrange	clrOrange	clrGold	clrYellow	clrChartreuse	clrLime	clrSpringGreen	clrAqua
clrDeepSkyBlue	clrBlue	clrMagenta	clrRed	clrGray	clrSlateGray	clrPeru	clrBlueViolet
clrLightSlateGray	clrDeepPink	clrMediumTurquoise	clrDodgerBlue	clrTurquoise	clrRoyalBlue	clrSlateBlue	clrDarkKhaki
clrIndianRed	clrMediumOrchid	clrGreenYellow	clrMediumAquamarine	clrDarkSeaGreen	clrTomato	clrRosyBrown	clrOrchid
clrMediumPurple	clrPaleVioletRed	clrCoral	clrCornflowerBlue	clrDarkGray	clrSandyBrown	clrMediumSlateBlue	clrTan
clrDarkSalmon	clrBurlyWood	clrHotPink	clrSalmon	clrViolet	clrLightCoral	clrSkyBlue	clrLightSalmon
clrPlum	clrKhaki	clrLightGreen	clrAquamarine	clrSilver	clrLightSkyBlue	clrLightSteelBlue	clrLightBlue
clrPaleGreen	clrThistle	clrPowderBlue	clrPaleGoldenrod	clrPaleTurquoise	clrLightGray	clrWheat	clrNavajoWhite
clrMoccasin	clrLightPink	clrGainsboro	clrPeachPuff	clrPink	clrBisque	clrLightGoldenrod	clrBlanchedAlmond
clrLemonChiffon	clrBeige	clrAntiqueWhite	clrPapayaWhip	clrCornsilk	clrLightYellow	clrLightCyan	clrLinen
clrLavender	clrMistyRose	clrOldLace	clrWhiteSmoke	clrSeashell	clrIvory	clrHoneydew	clrAliceBlue
clrLavenderBlush	clrMintCream	clrSnow	clrWhite				

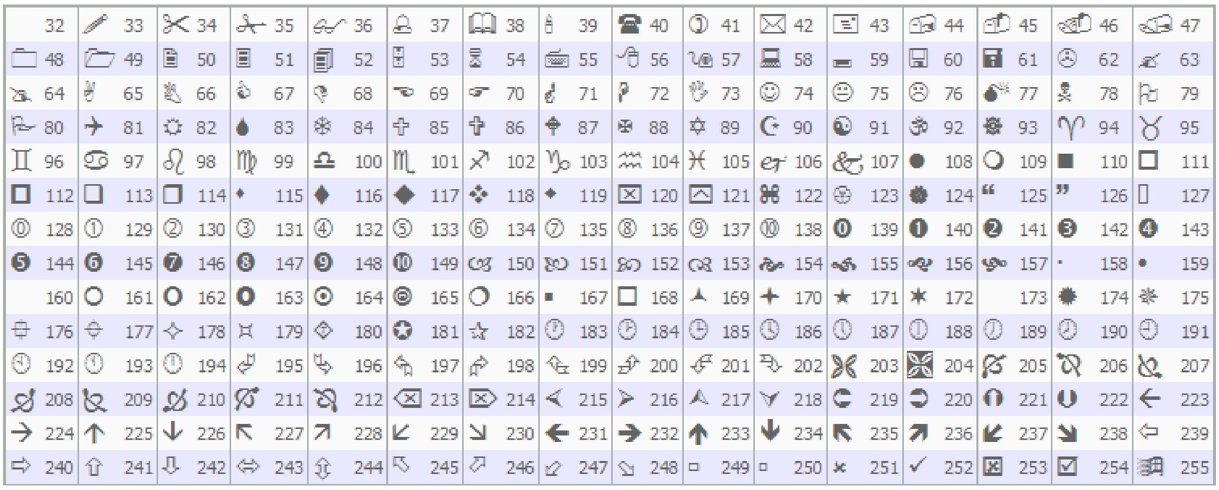
Color can be set to an object using the ObjectSetInteger() function. For setting color to custom indicators the PlotIndexSetInteger() function is used. For getting color values there are similar functions ObjectGetInteger() and PlotIndexGetInteger().

Example:

```
//---- indicator settings
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_type3   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_color2  clrRed
#property indicator_color3  clrLime
```

Wingdings

Characters of Wingdings used with the OBJ_ARROW object:



32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

A necessary character can be set using the ObjectSetInteger() function.

Example:

```
void OnStart()
  {
//---
   string up_arrow="up_arrow";
   datetime time=TimeCurrent();
   double lastClose[1];
   int close=CopyClose(Symbol(),Period(),0,1,lastClose);     // Get the Close price
//--- If the price was obtained
   if(close>0)
     {
      ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0,0);          // Create an arrow
      ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);    // Set the arrow code
      ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);        // Set time
      ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);// Set price
      ChartRedraw(0);                                        // Draw arrow now
     }
   else
      Print("Unable to get the latest Close price!");
  }
```

Indicators Constants

There are 37 predefined technical indicators, which can be used in programs written in the MQL5 language. In addition, there is an opportunity to create custom indicators using the iCustom() function. All constants required for that are divided into 5 groups:

- Price constants – for selecting the type of price or volume, on which an indicator is calculated;
- Smoothing methods – built-in smoothing methods used in indicators;
- Indicator lines – identifiers of indicator buffers when accessing indicator values using CopyBuffer();
- Drawing styles – for indicating one of 18 types of drawing and setting the line drawing style;
- Custom indicators properties are used in functions for working with custom indicators;
- Types of indicators are used for specifying the type of technical indicator when creating a handle using IndicatorCreate();
- Identifiers of data types are used for specifying the type of data passed in an array of the MqlParam type into the IndicatorCreate() function.

Price Constants

Calculations of technical indicators require price values and/or values of volumes, on which calculations will be performed. There are 7 predefined identifiers from the ENUM_APPLIED_PRICE enumeration, used to specify the desired price base for calculations.

ENUM_APPLIED_PRICE

ID	Description
PRICE_CLOSE	Close price
PRICE_OPEN	Open price
PRICE_HIGH	The maximum price for the period
PRICE_LOW	The minimum price for the period
PRICE_MEDIAN	Median price, (high + low)/2
PRICE_TYPICAL	Typical price, (high + low + close)/3
PRICE_WEIGHTED	Average price, (high + low + close + close)/4

If the volume is used in calculations, it's necessary to specify one of the two values from the ENUM_APPLIED_VOLUME enumeration.

ENUM_APPLIED_VOLUME

ID	Description
VOLUME_TICK	Tick volume
VOLUME_REAL	Trade volume

The iStochastic() technical Indicator can be calculated in two ways using:

- either only Close prices;
- or High and Low prices.

To select a necessary variant for calculation, specify one of the values of the ENUM_STO_PRICE enumeration.

ENUM_STO_PRICE

ID	Description
STO_LOWHIGH	Calculation is based on Low/High prices
STO_CLOSECLOSE	Calculation is based on Close/Close prices

If a technical indicator uses for calculations price data, type of which is set by ENUM_APPLIED_PRICE, then handle of any indicator (built in the terminal or written by a user) can be used as the input price series. In this case, values of the zero buffer of the indicator will be used for calculations. This makes it easy to build values of one indicator using values of another indicator. The handle of a custom indicator is created by calling the iCustom() function.

Example:

```
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- input parameters
input int      RSIperiod=14;         // Period for calculating the RSI
input int      Smooth=8;             // Smoothing period RSI
input ENUM_MA_METHOD meth=MODE_SMMA; // Method of smoothing
//---- plot RSI
#property indicator_label1  "RSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//---- plot RSI_Smoothed
#property indicator_label2  "RSI_Smoothed"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrNavy
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- indicator buffers
double         RSIBuffer[];          // Here we store the values of RSI
double         RSI_SmoothedBuffer[]; // Here will be smoothed values of RSI
int            RSIhandle;            // Handle to the RSI indicator
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,RSIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,RSI_SmoothedBuffer,INDICATOR_DATA);
   IndicatorSetString(INDICATOR_SHORTNAME,"iRSI");
   IndicatorSetInteger(INDICATOR_DIGITS,2);
//---
   RSIhandle=iRSI(NULL,0,RSIperiod,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                 const int prev_calculated,
                 const int begin,
                 const double &price[]
                 )

  {
```

```
//---  Reset the value of the last error
   ResetLastError();
//--- Get RSI indicator data in an array RSIBuffer []
   int copied=CopyBuffer(RSIhandle,0,0,rates_total,RSIBuffer);
   if(copied<=0)
     {
      Print("Unable to copy the values of the indicator RSI. Error = ",
            GetLastError(),",  copied =",copied);
      return(0);
     }
//--- Create the indicator of average values using values of RSI
   int RSI_MA_handle=iMA(NULL,0,Smooth,0,meth,RSIhandle);
   copied=CopyBuffer(RSI_MA_handle,0,0,rates_total,RSI_SmoothedBuffer);
   if(copied<=0)
     {
      Print("Unable to copy the smoothed indicator of RSI. Error = ",
            GetLastError(),",  copied =",copied);
      return(0);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Smoothing Methods

Many technical indicators are based on various methods of the price series smoothing. Some standard technical indicators require specification of the smoothing type as an input parameter. For specifying the desired type of smoothing, identifiers listed in the ENUM_MA_METHOD enumeration are used.

ENUM_MA_METHOD

ID	Description
MODE_SMA	Simple averaging
MODE_EMA	Exponential averaging
MODE_SMMA	Smoothed averaging
MODE_LWMA	Linear-weighted averaging

Example:

```
double ExtJaws[];
double ExtTeeth[];
double ExtLips[];
//---- handles for moving averages
int    ExtJawsHandle;
int    ExtTeethHandle;
int    ExtLipsHandle;
//--- get MA's handles
ExtJawsHandle=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtTeethHandle=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtLipsHandle=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
```

Indicators Lines

Some technical indicators have several buffers drawn in the chart. Numbering of indicator buffers starts with 0. When copying indicator values using the CopyBuffer() function into an array of the double type, for some indicators one may indicate the identifier of a copied buffer instead of its number.

Identifiers of indicator lines permissible when copying values of iMACD(), iRVI() and iStochastic().

Constant	Value	Description
MAIN_LINE	0	Main line
SIGNAL_LINE	1	Signal line

Identifiers of indicator lines permissible when copying values of ADX() and ADXW().

Constant	Value	Description
MAIN_LINE	0	Main line
PLUSDI_LINE	1	Line +DI
MINUSDI_LINE	2	Line -DI

Identifiers of indicator lines permissible when copying values of iBands().

Constant	Value	Description
BASE_LINE	0	Main line
UPPER_BAND	1	Upper limit
LOWER_BAND	2	Lower limit

Identifiers of indicator lines permissible when copying values of iEnvelopes() and iFractals().

Constant	Value	Description
UPPER_LINE	0	Upper line
LOWER_LINE	1	Bottom line

Identifiers of indicator lines permissible when copying values of iGator()

Constant	Value	Description
UPPER_HISTOGRAM	0	Upper histogram
LOWER_HISTOGRAM	2	Bottom histogram

Identifiers of indicator lines permissible when copying values of iAlligator().

Constant	Value	Description
GATORJAW_LINE	0	Jaw line
GATORTEETH_LINE	1	Teeth line
GATORLIPS_LINE	2	Lips line

Identifiers of indicator lines permissible when copying values of iIchimoku().

Constant	Value	Description
TENKANSEN_LINE	0	Tenkan-sen line
KIJUNSEN_LINE	1	Kijun-sen line
SENKOUSPANA_LINE	2	Senkou Span A line
SENKOUSPANB_LINE	3	Senkou Span B line
CHIKOUSPAN_LINE	4	Chikou Span line

Drawing Styles

When creating a custom indicator, you can specify one of 18 types of graphical plotting (as displayed in the main chart window or a chart subwindow), whose values are specified in the ENUM_DRAW_TYPE enumeration.

In one custom indicator, it is permissible to use any indicator building/drawing types. Each construction type requires specification of one to five global arrays for storing data necessary for drawing. These data arrays must be bound with indicator buffers using the SetIndexBuffer() function. The type of data from ENUM_INDEXBUFFER_TYPE should be specified for each buffer.

Depending on the drawing style, you may need one to four value buffers (marked as INDICATOR_DATA). If a style admits dynamic alternation of colors (all styles contain COLOR in their names), then you'll need one more buffer of color (indicated type INDICATOR_COLOR_INDEX). The color buffers are always bound after value buffers corresponding to the style.

ENUM_DRAW_TYPE

ID	Description	Data buffers	Color buffers
DRAW_NONE	Not drawn	1	0
DRAW_LINE	Line	1	0
DRAW_SECTION	Section	1	0
DRAW_HISTOGRAM	Histogram from the zero line	1	0
DRAW_HISTOGRAM2	Histogram of the two indicator buffers	2	0
DRAW_ARROW	Drawing arrows	1	0
DRAW_ZIGZAG	Style Zigzag allows vertical section on the bar	2	0
DRAW_FILLING	Color fill between the two levels	2	0

ID	Description	Data buffers	Color buffers
DRAW_BARS	Display as a sequence of bars	4	0
DRAW_CANDLES	Display as a sequence of candlesticks	4	0
DRAW_COLOR_LINE	Multicolored line	1	1
DRAW_COLOR_SECTION	Multicolored section	1	1
DRAW_COLOR_HISTOGRAM	Multicolored histogram from the zero line	1	1
DRAW_COLOR_HISTOGRAM2	Multicolored histogram of the two indicator buffers	2	1
DRAW_COLOR_ARROW	Drawing multicolored arrows	1	1
DRAW_COLOR_ZIGZAG	Multicolored ZigZag	2	1
DRAW_COLOR_BARS	Multicolored bars	4	1
DRAW_COLOR_CANDLES	Multicolored candlesticks	4	1

To refine the display of the selected drawing type identifiers listed in ENUM_PLOT_PROPERTY are used.

For functions PlotIndexSetInteger() and PlotIndexGetInteger()

ENUM_PLOT_PROPERTY_INTEGER

ID	Description	Property type
PLOT_ARROW	Arrow code for style DRAW_ARROW	uchar
PLOT_ARROW_SHIFT	Vertical shift of arrows for style DRAW_ARROW	int
PLOT_DRAW_BEGIN	Number of initial bars without drawing and values in the DataWindow	int
PLOT_DRAW_TYPE	Type of graphical construction	ENUM_DRAW_TYPE
PLOT_SHOW_DATA	Sign of display of construction values in the DataWindow	bool
PLOT_SHIFT	Shift of indicator plotting along the time axis in bars	int
PLOT_LINE_STYLE	Drawing line style	ENUM_LINE_STYLE
PLOT_LINE_WIDTH	The thickness of the drawing line	int
PLOT_COLOR_INDEXES	The number of colors	int
PLOT_LINE_COLOR	The index of a buffer containing the drawing color	color modifier = index number of colors

For the function PlotIndexSetDouble()

ENUM_PLOT_PROPERTY_DOUBLE

ID	Description	Property type
PLOT_EMPTY_VALUE	An empty value for plotting, for	double

ID	Description	Property type
	which there is no drawing	

For the function PlotIndexSetString()

ENUM_PLOT_PROPERTY_STRING

ID	Description	Property type
PLOT_LABEL	The name of the indicator graphical series to display in the DataWindow. When working with complex graphical styles requiring several indicator buffers for display, the names for each buffer can be specified using ";" as a separator. Sample code is shown in DRAW_CANDLES	string

5 styles can be used for drawing lines in custom indicators. They are valid only for the line thickness 0 or 1.

ENUM_LINE_STYLE

ID	Description
STYLE_SOLID	Solid line
STYLE_DASH	Broken line
STYLE_DOT	Dotted line
STYLE_DASHDOT	Dash-dot line
STYLE_DASHDOTDOT	Dash - two points

To set the line drawing style and the type of drawing, the PlotIndexSetInteger() function is used. For the Fibonacci extensions the thickness and drawing style of levels can be indicated using the ObjectSetInteger() function.

Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- indicator buffers
double         MABuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Bind the Array to the indicator buffer with index 0
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
//--- Set the line drawing
   PlotIndexSetInteger(0,PLOT_DRAW_TYPE,DRAW_LINE);
//--- Set the style line
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,STYLE_DOT);
//--- Set line color
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrRed);
//--- Set line thickness
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
//--- Set labels for the line
   PlotIndexSetString(0,PLOT_LABEL,"Moving Average");
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   for(int i=prev_calculated;i<rates_total;i++)
     {
      MABuffer[i]=close[i];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Custom Indicators Properties

The number of indicator buffers that can be used in a custom indicator is unlimited. But for each array, which is designated as the indicator buffer using the SetIndexBuffer() function, it's necessary to specify the data type that it will store. This may be one of the values of the ENUM_INDEXBUFFER_TYPE enumeration.

ENUM_INDEXBUFFER_TYPE

ID	Description
INDICATOR_DATA	Data to draw
INDICATOR_COLOR_INDEX	Color
INDICATOR_CALCULATIONS	Auxiliary buffers for intermediate calculations

A custom indicator has a lot of settings to provide convenient displaying. These settings are made through the assignment of corresponding indicator properties using functions IndicatorSetDouble(), IndicatorSetInteger() and IndicatorSetString(). Identifiers of indicator properties are listed in the ENUM_CUSTOMIND_PROPERTY enumeration.

ENUM_CUSTOMIND_PROPERTY_INTEGER

ID	Description	Property type
INDICATOR_DIGITS	Accuracy of drawing of indicator values	int
INDICATOR_HEIGHT	Fixed height of the indicator's window (the prep	int

ID	Description	Property type
	rocessor command #property indicator_height)	
INDICATOR_LEVELS	Number of levels in the indicator window	int
INDICATOR_LEVELCOLOR	Color of the level line	color modifier = level number
INDICATOR_LEVELSTYLE	Style of the level line	ENUM_LINE_STYLE modifier = level number
INDICATOR_LEVELWIDTH	Thickness of the level line	int modifier = level number

ID	Description	Property type
INDICATOR_FIXED_MINIMUM	Fixed minimum for the indicator window. The property can only be written by the IndicatorSetInteger() function	bool
INDICATOR_FIXED_MAXIMUM	Fixed maximum for the indicator window. The property	bool

ID	Description	Property type
	can only be written by the IndicatorSetInteger() function	

ENUM_CUSTOMIND_PROPERTY_DOUBLE

ID	Description	Property type
INDICATOR_MINIMUM	Minimum of the indicator window	double
INDICATOR_MAXIMUM	Maximum of the indicator window	double
INDICATOR_LEVELVALUE	Level value	double modifier = level number

ENUM_CUSTOMIND_PROPERTY_STRING

ID	Description	Property type
INDICATOR_SHORTNAME	Short indicator name	string
INDICATOR_LEVELTEXT	Level description	string modifier = level number

Examples:

```
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_color2  clrRed
//--- input parameters
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//--- indicator buffers
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,MainBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,HighesBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,LowesBuffer,INDICATOR_CALCULATIONS);
//--- set accuracy
   IndicatorSetInteger(INDICATOR_DIGITS,2);
```

```
//--- set levels
   IndicatorSetInteger(INDICATOR_LEVELS,2);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,0,20);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,1,80);
//--- set maximum and minimum for subwindow
   IndicatorSetDouble(INDICATOR_MINIMUM,0);
   IndicatorSetDouble(INDICATOR_MAXIMUM,100);
//--- sets first bar from which index will be drawn
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,KPeriod+Slowing-2);
   PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,KPeriod+Slowing+DPeriod);
//--- set style STYLE_DOT for second line
   PlotIndexSetInteger(1,PLOT_LINE_STYLE,STYLE_DOT);
//--- name for DataWindow and indicator subwindow label
   IndicatorSetString(INDICATOR_SHORTNAME,"Stoch("+KPeriod+","+DPeriod+","+Slowing+")"
   PlotIndexSetString(0,PLOT_LABEL,"Main");
   PlotIndexSetString(1,PLOT_LABEL,"Signal");
//--- sets drawing line to empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0.0);
//--- initialization done
  }
```

Types of Technical Indicators

There are two ways to create an indicator handle for further accessing to its values. The first way is to directly specify a function name from the list of technical indicators. The second method using the IndicatorCreate() is to uniformly create a handle of any indicator by assigning an identifier from the ENUM_INDICATOR enumeration. Both ways of handle creation are equal, you can use the one that is most convenient in a particular case when writing a program in MQL5.

When creating an indicator of type IND_CUSTOM, the *type* field of the first element of an array of input parameters MqlParam must have the TYPE_STRING value of the enumeration ENUM_DATATYPE, while the field *string_value* of the first element must contain the name of the custom indicator.

ENUM_INDICATOR

Identifier	Indicator
IND_AC	Accelerator Oscillator
IND_AD	Accumulation/Distribution
IND_ADX	Average Directional Index
IND_ADXW	ADX by Welles Wilder
IND_ALLIGATOR	Alligator
IND_AMA	Adaptive Moving Average
IND_AO	Awesome Oscillator
IND_ATR	Average True Range
IND_BANDS	Bollinger Bands®
IND_BEARS	Bears Power
IND_BULLS	Bulls Power
IND_BWMFI	Market Facilitation Index
IND_CCI	Commodity Channel Index
IND_CHAIKIN	Chaikin Oscillator
IND_CUSTOM	Custom indicator
IND_DEMA	Double Exponential Moving Average
IND_DEMARKER	DeMarker
IND_ENVELOPES	Envelopes
IND_FORCE	Force Index
IND_FRACTALS	Fractals
IND_FRAMA	Fractal Adaptive Moving Average
IND_GATOR	Gator Oscillator

Identifier	Indicator
IND_ICHIMOKU	Ichimoku Kinko Hyo
IND_MA	Moving Average
IND_MACD	MACD
IND_MFI	Money Flow Index
IND_MOMENTUM	Momentum
IND_OBV	On Balance Volume
IND_OSMA	OsMA
IND_RSI	Relative Strength Index
IND_RVI	Relative Vigor Index
IND_SAR	Parabolic SAR
IND_STDDEV	Standard Deviation
IND_STOCHASTIC	Stochastic Oscillator
IND_TEMA	Triple Exponential Moving Average
IND_TRIX	Triple Exponential Moving Averages Oscillator
IND_VIDYA	Variable Index Dynamic Average
IND_VOLUMES	Volumes
IND_WPR	Williams' Percent Range

Data Type Identifiers

When creating an indicator handle using the IndicatorCreate() function, an array of MqlParam type must be specified as the last parameter. Accordingly, the MqlParam structure, describing indicator, contains a special field *type*. This field contains information about the data type (real, integer or string type) that are passed by a particular element of the array. The value of this field of the MqlParam structure may be one of ENUM_DATATYPE values.

ENUM_DATATYPE

Identifier	Data type
TYPE_BOOL	bool
TYPE_CHAR	char
TYPE_UCHAR	uchar
TYPE_SHORT	short
TYPE_USHORT	ushort
TYPE_COLOR	color
TYPE_INT	int
TYPE_UINT	uint
TYPE_DATETIME	datetime
TYPE_LONG	long
TYPE_ULONG	ulong
TYPE_FLOAT	float
TYPE_DOUBLE	double
TYPE_STRING	string

Each element of the array describes the corresponding input parameter of a created technical indicator, so the type and order of elements in the array must be strictly maintained in accordance with the description.

Environment State

Constants describing the current runtime environment of an mql5-program are divided into groups:

- Client terminal properties – information about the client terminal;
- Executed MQL5-program properties – mql5 program properties, which help to control its execution;
- Symbol properties – obtaining information about a symbol;
- Account properties – information about the current account;
- Testing Statistics – results of Expert Advisor testing.

Client Terminal Properties

Information about the client terminal can be obtained by two functions: TerminalInfoInteger() and TerminalInfoString(). For parameters, these functions accept values from ENUM_TERMINAL_INFO_INTEGER and ENUM_TERMINAL_INFO_STRING respectively.

ENUM_TERMINAL_INFO_INTEGER

Identifier	Description	Type
TERMINAL_BUILD	The client terminal build number	int
TERMINAL_COMMUNITY_ACCOUNT	The flag indicates the presence of MQL5.community authorization data in the terminal	bool
TERMINAL_COMMUNITY_CONNECTION	Connection to MQL5.community	bool
TERMINAL_CONNECTED	Connection to a trade server	bool
TERMINAL_DLLS_ALLOWED	Permission to use DLL	bool
TERMINAL_TRADE_ALLOWED	Permission to trade	bool
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the terminal settings	bool
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	bool
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to smartphone	bool
TERMINAL_MAXBARS	The maximal bars count on the chart	int
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID data for Push notifications	bool
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	int
TERMINAL_CPU_CORES	The number of CPU cores in the system	int
TERMINAL_DISK_SPACE	Free disk space for the MQL5\Files folder of the terminal (agent), MB	int
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, MB	int
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal (agent), MB	int

Identifier	Description	Type
TERMINAL_MEMORY_AVAILABLE	Free memory of the terminal (agent) process, MB	int
TERMINAL_MEMORY_USED	Memory used by the terminal (agent), MB	int
TERMINAL_X64	Indication of the "64-bit terminal"	bool
TERMINAL_OPENCL_SUPPORT	The version of the supported OpenCL in the format of 0x00010002 = 1.2. "0" means that OpenCL is not supported	int
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI). Knowing the parameter value, you can set the size of graphical objects so that they look the same on monitors with different resolution characteristics.	int
TERMINAL_SCREEN_LEFT	The left coordinate of the virtual screen. A virtual screen is a rectangle that covers all monitors. If the system has two monitors ordered from right to left, then the left coordinate of the virtual screen can be on the border of two monitors.	int
TERMINAL_SCREEN_TOP	The top coordinate of the virtual screen	int
TERMINAL_SCREEN_WIDTH	Terminal width	int
TERMINAL_SCREEN_HEIGHT	Terminal height	int
TERMINAL_LEFT	The left coordinate of the terminal relative to the virtual screen	int
TERMINAL_TOP	The top coordinate of the terminal relative to the virtual screen	int
TERMINAL_RIGHT	The right coordinate of the terminal relative to the virtual screen	int
TERMINAL_BOTTOM	The bottom coordinate of the terminal relative to the virtual screen	int

Identifier	Description	Type
TERMINAL_PING_LAST	The last known value of a ping to a trade server in microseconds. One second comprises of one million microseconds	int
TERMINAL_VPS	Indication that the terminal is launched on the MetaTrader Virtual Hosting server (MetaTrader VPS)	bool
Key identifier	**Description**	
TERMINAL_KEYSTATE_LEFT	State of the "Left arrow" key	int
TERMINAL_KEYSTATE_UP	State of the "Up arrow" key	int
TERMINAL_KEYSTATE_RIGHT	State of the "Right arrow" key	int
TERMINAL_KEYSTATE_DOWN	State of the "Down arrow" key	int
TERMINAL_KEYSTATE_SHIFT	State of the "Shift" key	int
TERMINAL_KEYSTATE_CONTROL	State of the "Ctrl" key	int
TERMINAL_KEYSTATE_MENU	State of the "Windows" key	int
TERMINAL_KEYSTATE_CAPSLOCK	State of the "CapsLock" key	int
TERMINAL_KEYSTATE_NUMLOCK	State of the "NumLock" key	int
TERMINAL_KEYSTATE_SCRLOCK	State of the "ScrollLock" key	int
TERMINAL_KEYSTATE_ENTER	State of the "Enter" key	int
TERMINAL_KEYSTATE_INSERT	State of the "Insert" key	int
TERMINAL_KEYSTATE_DELETE	State of the "Delete" key	int
TERMINAL_KEYSTATE_HOME	State of the "Home" key	int
TERMINAL_KEYSTATE_END	State of the "End" key	int
TERMINAL_KEYSTATE_TAB	State of the "Tab" key	int
TERMINAL_KEYSTATE_PAGEUP	State of the "PageUp" key	int
TERMINAL_KEYSTATE_PAGEDOWN	State of the "PageDown" key	int
TERMINAL_KEYSTATE_ESCAPE	State of the "Escape" key	int

Call to TerminalInfoInteger(TERMINAL_KEYSTATE_XXX) returns the same state code of a key as the GetKeyState() function in MSDN.

Example of scaling factor calculation:

```
//--- Creating a 1.5 inch wide button on a screen
int screen_dpi = TerminalInfoInteger(TERMINAL_SCREEN_DPI); // Find DPI of the user mor
int base_width = 144;                                    // The basic width in the
int width      = (button_width * screen_dpi) / 96;       // Calculate the button wi
...

//--- Calculating the scaling factor as a percentage
int scale_factor=(TerminalInfoInteger(TERMINAL_SCREEN_DPI) * 100) / 96;
//--- Use of the scaling factor
width=(base_width * scale_factor) / 100;
```

In the above example, the graphical resource looks the same on monitors with different resolution characteristics. The size of control elements (buttons, dialog windows, etc.) corresponds to personalization settings.

ENUM_TERMINAL_INFO_DOUBLE

Identifier	Description	Type
TERMINAL_COMMUNITY_BALANCE	Balance in MQL5.community	double
TERMINAL_RETRANSMISSION	Percentage of resent network packets in the TCP/IP protocol for all running applications and services on the given computer. Packet loss occurs even in the fastest and correctly configured networks. In this case, there is no confirmation of packet delivery between the recipient and the sender, therefore lost packets are resent. It is not an indication of the connection quality between a particular terminal and a trade server, since the percentage is calculated for the entire network activity, including system and background activity. The TERMINAL_RETRANSMISSION value is requested from the operating system once per minute. The terminal itself does not calculate this value.	double

File operations can be performed only in two directories; corresponding paths can be obtained using the request for TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH properties.

ENUM_TERMINAL_INFO_STRING

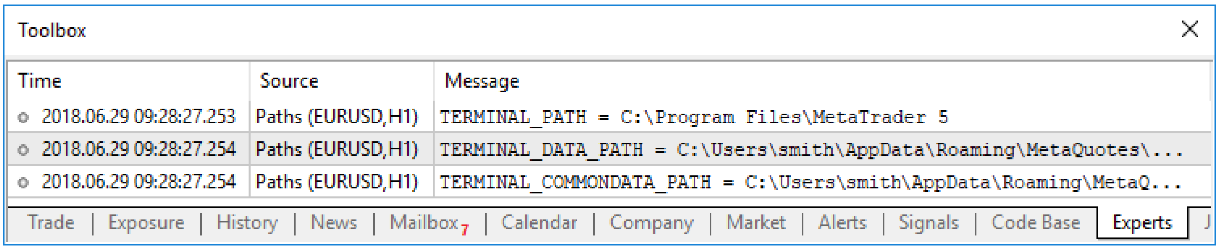
Identifier	Description	Type
TERMINAL_LANGUAGE	Language of the terminal	string
TERMINAL_COMPANY	Company name	string
TERMINAL_NAME	Terminal name	string
TERMINAL_PATH	Folder from which the terminal is started	string
TERMINAL_DATA_PATH	Folder in which terminal data are stored	string
TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on a computer	string
TERMINAL_CPU_NAME	CPU name	string
TERMINAL_CPU_ARCHITECTURE	CPU architecture	string
TERMINAL_OS_VERSION	User's OS name	string
TERMINAL_COLORTHEME_NAME	Terminal color scheme; possible values: Light and Dark.	string

For a better understanding of paths, stored in properties of TERMINAL_PATH, TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH parameters, it is recommended to execute the script, which will return these values for the current copy of the client terminal, installed on your computer

Example: Script returns information about the client terminal paths

```
//+------------------------------------------------------------------+
//|                                          Check_TerminalPaths.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   Print("TERMINAL_PATH = ",TerminalInfoString(TERMINAL_PATH));
   Print("TERMINAL_DATA_PATH = ",TerminalInfoString(TERMINAL_DATA_PATH));
   Print("TERMINAL_COMMONDATA_PATH = ",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
  }
```

As result of the script execution in the Experts Journal you will see a messages, like the following:



```
Time                     Source            Message
2018.06.29 09:28:27.253  Paths (EURUSD,H1) TERMINAL_PATH = C:\Program Files\MetaTrader 5
2018.06.29 09:28:27.254  Paths (EURUSD,H1) TERMINAL_DATA_PATH = C:\Users\smith\AppData\Roaming\MetaQuotes\...
2018.06.29 09:28:27.254  Paths (EURUSD,H1) TERMINAL_COMMONDATA_PATH = C:\Users\smith\AppData\Roaming\MetaQ...
```

Get Terminal Color Scheme Information

The terminal supports two color schemes: Light (default) and Dark. When developing custom applications with graphical user interfaces, programmers should take the current terminal color scheme into account. Visual components used in the application should be dynamically adaptable to enhance user experience and maintain visual consistency.

To support this, the language provides functions for detecting the terminal color scheme:

- The TERMINAL_COLORTHEME_NAME value from the ENUM_TERMINAL_INFO_STRING enumeration allows you to retrieve the name of the current color scheme using the TerminalInfoString function. Possible values: Light and Dark.
- Use the THEME_COLOR_* values from the ENUM_TERMINAL_INFO_INTEGER enumeration to retrieve the colors of specific UI elements through the TerminalInfoInteger function.

Identifier	Description	Property type
THEME_COLOR_WINDOW	Window background	color
THEME_COLOR_WINDOWTEXT	Text in the window	color
THEME_COLOR_BTNTEXT	Button text	color
THEME_COLOR_GRAYTEXT	Inactive (disabled) text	color
THEME_COLOR_INFOTEXT	Tooltip text	color
THEME_COLOR_INFOBK	Tooltip background	color
THEME_COLOR_3DFACE	Front face of 3D elements	color
THEME_COLOR_3DLIGHT	Light side of 3D elements	color
THEME_COLOR_3DSHADOW	Shadow side of 3D elements	color
THEME_COLOR_3DDKSHADOW	Dark shadow of 3D elements	color
THEME_COLOR_3DHILIGHT	Highlight of 3D elements	color
THEME_COLOR_HIGHLIGHT	Background of selected elements	color
THEME_COLOR_HIGHLIGHTTEXT	Text of selected elements	color

Identifier	Description	Property type
THEME_COLOR_BTNFACE	Button front face	color
THEME_COLOR_BTNHILIGHT	Button highlight	color
THEME_COLOR_BTNSHADOW	Button shadow	color
THEME_COLOR_MENU	Menu background	color
THEME_COLOR_MENUBAR	Menu bar background	color
THEME_COLOR_MENUTEXT	Menu text	color
THEME_COLOR_MENUHILIGHT	Highlight of selected menu item	color
THEME_COLOR_ACTIVECAPTION	Active window title	color
THEME_COLOR_INACTIVECAPTION	Inactive window title	color
THEME_COLOR_GRADIENTINACTIVECAPTION	Gradient of inactive window title	color
THEME_COLOR_CAPTIONTEXT	Window title text	color
THEME_COLOR_INACTIVECAPTIONTEXT	Inactive window title text	color
THEME_COLOR_HOTTEXT	Hyperlinks or active elements	color
THEME_COLOR_NONE	Color not selected	color
THEME_COLOR_SEPARATOR	Separator	color
THEME_COLOR_SCROLLBACK	Scrollbar	color
THEME_COLOR_LINE1	Background color of odd rows in the Journal	color
THEME_COLOR_LINE2	Background color of even rows in the Journal	color
THEME_COLOR_GRID	Grid color in the Journal	color
THEME_COLOR_SUMMARY	Background color of summary row in the Journal	color
THEME_COLOR_ERROR	Error message text color	color
THEME_COLOR_INVALID	Invalid value text color	color
THEME_COLOR_NEGATIVE	Negative value color	color
THEME_COLOR_POSITIVE	Positive value color	color
THEME_COLOR_LINK	Link color	color
THEME_COLOR_LINKHOVER	Link hover color	color
THEME_COLOR_LINKTESTER	Link color from cached results of previous testing/optimization runs	color
THEME_COLOR_TEXTUP	"Button released" state	color

Identifier	Description	Property type
THEME_COLOR_TEXTDOWN	"Button pressed" state	color
THEME_COLOR_BACKUP	Color of "BUY" and "SELL" buttons in "One Click Trading" when the quote increases	color
THEME_COLOR_BACKDOWN	Color of "BUY" and "SELL" buttons in "One Click Trading" when the quote decreases	color
THEME_COLOR_CLOSE	"Close" operation button color	color
THEME_COLOR_BUY	"BUY" operation button color	color
THEME_COLOR_SELL	"SELL" operation button color	color
THEME_COLOR_DEPOSIT	"Deposit" button color	color
THEME_COLOR_WITHDRAWAL	"Withdrawal" button color	color
THEME_COLOR_BID	Bid line color	color
THEME_COLOR_ASK	Ask line color	color
THEME_COLOR_STOPS	Stop line color	color
THEME_COLOR_STOPS_RED	StopLoss value highlight when profit is negative (Trade tab)	color
THEME_COLOR_STOPS_GREEN	StopLoss value highlight when profit is positive (Trade tab)	color
THEME_COLOR_CONFIRM	"Accept" button color in the order submission window	color
THEME_COLOR_REQUOTE	"Requote" button color in the order submission window	color
THEME_COLOR_REJECT	"Reject" button color in the order submission window	color
THEME_COLOR_NOTIFICATION	Color of a change notification from the server in the order submission window	color
THEME_COLOR_RATING	Rating bar color in the learning system	color
THEME_COLOR_BOOK_BUY	Background color of buy levels in the Depth of Market	color
THEME_COLOR_BOOK_SELL	Background color of sell levels in the Depth of Market	color
THEME_COLOR_BOOK_LAST	Color of the last trade in the Depth of Market	color

Identifier	Description	Property type
THEME_COLOR_BOOK_STOP	StopLoss level color in the Depth of Market	color
THEME_COLOR_BOOK_SPREAD	Background color of levels within the spread in the Depth of Market	color
THEME_COLOR_TICKS_BID	Bid line color on the tick chart in the order submission window	color
THEME_COLOR_TICKS_ASK	Ask line color on the tick chart in the order submission window	color
THEME_COLOR_TICKS_LAST	Last line color on the tick chart in the order submission window	color
THEME_COLOR_TICKS_CROSS	Crosshair color on the tick chart in the order submission window	color
THEME_COLOR_TICKS_SL	StopLoss line color on the tick chart in the order submission window	color
THEME_COLOR_TICKS_TP	TakeProfit line color on the tick chart in the order submission window	color
THEME_COLOR_TESTER_START	"Start" button color in testing/optimization	color
THEME_COLOR_TESTER_STOP	"Stop" button color in testing/optimization	color
THEME_COLOR_TESTER_START_FRAME	Border color of the "Start" button in testing/optimization	color
THEME_COLOR_TESTER_STOP_FRAME	Border color of the "Stop" button in testing/optimization	color
THEME_COLOR_TESTER_PROGRESS	Progress bar color in testing/optimization	color
THEME_COLOR_TESTER_BALANCE	Balance line color in the Strategy Tester	color
THEME_COLOR_TESTER_EQUITY	Equity line color in the Strategy Tester	color
THEME_COLOR_TESTER_MARGIN	Deposit Load graph color in the Strategy Tester	color
THEME_COLOR_PROFILER_CALL	Color of the code line with a call during Profiling	color
THEME_COLOR_PROFILER_CALLSEL	Selected code line color with a call during Profiling	color

Identifier	Description	Property type
THEME_COLOR_PROFILER_LINE	Line color in the Profiler Journal	color
THEME_COLOR_PROFILER_LINESEL	Selected line color in the Profiler Journal	color

Running MQL5 Program Properties

To obtain information about the currently running mql5 program, constants from ENUM_MQL_INFO_INTEGER and ENUM_MQL_INFO_STRING are used.

For function MQLInfoInteger

ENUM_MQL_INFO_INTEGER

Identifier	Description	Type
MQL_HANDLES_USED	The current number of active object handles. These include both dynamic (created via new) and non-dynamic objects, global/local variables or class members. The more handles a program uses, the more resources it consumes.	int
MQL_MEMORY_LIMIT	Maximum possible amount of dynamic memory for MQL5 program in MB	int
MQL_MEMORY_USED	Memory used by MQL5 program in MB	int
MQL_PROGRAM_TYPE	Type of the MQL5 program	ENUM_PROGRAM_TYPE
MQL_DLLS_ALLOWED	The permission to use DLL **for the given running program**	bool
MQL_TRADE_ALLOWED	The permission to trade **for the given running program**	bool
MQL_SIGNALS_ALLOWED	The permission to modify the Signals **for the**	bool

Identifier	Description	Type
	given running program	
MQL_DEBUG	Indication that the program is running in the debugging mode	bool
MQL_PROFILER	Indication that the program is running in the code profiling mode	bool
MQL_TESTER	Indication that the program is running in the tester	bool
MQL_FORWARD	Indication that the program is running in the forward testing process	bool
MQL_OPTIMIZATION	Indication that the program is running in the optimization mode	bool
MQL_VISUAL_MODE	Indication that the program is running in the visual testing mode	bool
MQL_FRAME_MODE	Indication that the Expert Advisor is running in gathering optimization result frames mode	bool
MQL_LICENSE_TYPE	Type of license of the EX5 module. The license refers to the EX5 module, from which a request is made using	ENUM_LICENSE_TYPE

Identifier	Description	Type
	MQLInfoInteger(MQL_LICENSE_TYPE).	
MQL_STARTED_FROM_CONFIG	Returns true if a script/EA is launched from the StartUp section of the configuration file. This means that the script/EA had been specified in the configuration file the terminal was launched with.	bool

For function MQLInfoString

ENUM_MQL_INFO_STRING

Identifier	Description	Type
MQL_PROGRAM_NAME	Name of the running mql5-program	string
MQL5_PROGRAM_PATH	Path **for the given running program**	string

For information about the type of the running program, values of ENUM_PROGRAM_TYPE are used.

ENUM_PROGRAM_TYPE

Identifier	Description
PROGRAM_SCRIPT	Script
PROGRAM_EXPERT	Expert
PROGRAM_INDICATOR	Indicator
PROGRAM_SERVICE	Service

ENUM_LICENSE_TYPE

Identifier	Description
LICENSE_FREE	A free unlimited version

Identifier	Description
LICENSE_DEMO	A trial version of a paid product from the Market. It works only in the strategy tester
LICENSE_FULL	A purchased licensed version allows at least 5 activations. The number of activations is specified by seller. Seller may increase the allowed number of activations
LICENSE_TIME	A version with a limited term license

Example:

```
   ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PROGRAM_TYPE);
   switch(mql_program)
     {
      case PROGRAM_SCRIPT:
        {
         Print(__FILE__+" is script");
         break;
        }
      case PROGRAM_EXPERT:
        {
         Print(__FILE__+" is Expert Advisor");
         break;
        }
      case PROGRAM_INDICATOR:
        {
         Print(__FILE__+" is custom indicator");
         break;
        }
      default:Print("MQL5 type value is ",mql_program);
   //---
   Print("MQLInfoInteger(MQL_MEMORY_LIMIT)=", MQLInfoInteger(MQL_MEMORY_LIMIT), " MB")
   Print("MQLInfoInteger(MQL_MEMORY_USED)=", MQLInfoInteger(MQL_MEMORY_USED), " MB");
   Print("MQLInfoInteger(MQL_HANDLES_USED)=", MQLInfoInteger(MQL_HANDLES_USED), " han
```

Symbol Properties

To obtain the current market information there are several functions: SymbolInfoInteger(), SymbolInfoDouble() and SymbolInfoString(). The first parameter is the symbol name, the values of the second function parameter can be one of the identifiers of ENUM_SYMBOL_INFO_INTEGER, ENUM_SYMBOL_INFO_DOUBLE and ENUM_SYMBOL_INFO_STRING.

For function SymbolInfoInteger()

ENUM_SYMBOL_INFO_INTEGER

Identifier	Description	Type
SYMBOL_SUBSCRIPTION_DELAY	Symbol data arrives with a delay. The property can be requested only for symbols selected in MarketWatch (SYMBOL_SELECT = true).	bool

Identifier	Description	Type
	The ERR_MARKET_NOT_SELECTED (4302) error is generated for other symbols	
SYMBOL_SECTOR	The sector of the economy to which the asset belongs	ENUM_SYMBOL_SECTOR
SYMBOL_INDUSTRY	The industry or the economy bran	ENUM_SYMBOL_INDUSTRY

Identifier	Description	Type
	ch to which the symbol belongs	
SYMBOL_CUSTOM	It is a custom symbol - the symbol has been created synthetically based on other symbols from the Market Watch and/or external data	bool

Identifier	Description	Type
	sources	
SYMBOL_BACKGROUND_COLOR	The color of the background used for the symbol in Market Watch	color
SYMBOL_CHART_MODE	The price type used for generating symbols bars, i.e. Bid or Last	ENUM_SYMBOL_CHART_MODE
SYMBOL_EXIST	Symbol with this name	bool

Identifier	Description	Type
	exists	
SYMBOL_SELECT	Symbol is selected in Market Watch	bool
SYMBOL_VISIBLE	Symbol is visible in Market Watch. Some symbols (mostly, these are cross rates required for calculation of margin	bool

Identifier	Description	Type
	requirements or profits in deposit currency) are selected automatically, but may not be visible in Market Watch. To be displayed such symbols have to be explicitly sele	

Identifier	Description	Type
	cted.	
SYMBOL_SESSION_DEALS	Number of deals in the current session	long
SYMBOL_SESSION_BUY_ORDERS	Number of Buy orders at the moment	long
SYMBOL_SESSION_SELL_ORDERS	Number of Sell orders at the moment	long
SYMBOL_VOLUME	Volume of the last deal	long
SYMBOL_VOLUMEHIGH	Maximal day	long

Identifier	Description	Type
	volume	
SYMBOL_VOLUMELOW	Minimal day volume	long
SYMBOL_TIME	Time of the last quote	datetime
SYMBOL_TIME_MSC	Time of the last quote in milliseconds since 1970.01.01	long
SYMBOL_DIGITS	Digits after a decimal point	int
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	bool

Identifier	Description	Type
SYMBOL_SPREAD	Spread value in points	int
SYMBOL_TICKS_BOOKDEPTH	Maximal number of requests shown in Depth of Market. For symbols that have no queue of requests, the value is equal to zero.	int
SYMBOL_TRADE_CALC_MODE	Contract price	ENUM_SYMBOL_CALC_MODE

Identifier	Description	Type
	calculation mode	
SYMBOL_TRADE_MODE	Order execution type	ENUM_SYMBOL_TRADE_MODE
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	datetime
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	datetime

Identifier	Description	Type
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to place Stop orders	int
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	int
SYMBOL_TRADE_EXEMODE	Deal execution mode	ENUM_SYMBOL_TRADE_EXECUTION
SYMBOL_SWAP_MODE	Swap calculati	ENUM_SYMBOL_SWAP_MODE

Identifier	Description	Type
	on model	
SYMBOL_SWAP_ROLLOVER3DAYS	The day of week to charge 3-day swap rollover	ENUM_DAY_OF_WEEK
SYMBOL_MARGIN_HEDGED_USE_LEG	Calculating hedging margin using the larger leg (Buy or Sell)	bool
SYMBOL_EXPIRATION_MODE	Flags of allowed order expiration modes	int

Identifier	Description	Type
SYMBOL_FILLING_MODE	Flags of allowed order filling modes	int
SYMBOL_ORDER_MODE	Flags of allowed order types	int
SYMBOL_ORDER_GTC_MODE	Expiration of Stop Loss and Take Profit orders, if SYMBOL_EXPIRATION_MODE=SYMBOL_EXPIRATI	ENUM_SYMBOL_ORDER_GTC_MODE

Identifier	Description	Type
	ON_GTC (Good till canceled)	
SYMBOL_OPTION_MODE	Option type	ENUM_SYMBOL_OPTION_MODE
SYMBOL_OPTION_RIGHT	Option right (Call/Put)	ENUM_SYMBOL_OPTION_RIGHT

For function SymbolInfoDouble()

ENUM_SYMBOL_INFO_DOUBLE

Identifier	Description	Type
SYMBOL_BID	Bid - best sell offer	double
SYMBOL_BIDHIGH	Maximal Bid of the day	double
SYMBOL_BIDLOW	Minimal Bid of the day	double
SYMBOL_ASK	Ask - best buy offer	double
SYMBOL_ASKHIGH	Maximal Ask of the day	double

Identifier	Description	Type
SYMBOL_ASKLOW	Minimal Ask of the day	double
SYMBOL_LAST	Price of the last deal	double
SYMBOL_LASTHIGH	Maximal Last of the day	double
SYMBOL_LASTLOW	Minimal Last of the day	double
SYMBOL_VOLUME_REAL	Volume of the last deal	double
SYMBOL_VOLUMEHIGH_REAL	Maximum Volume of the day	double
SYMBOL_VOLUMELOW_REAL	Minimum Volume of the day	double
SYMBOL_OPTION_STRIKE	The strike price of an option. The price at which an option buyer can buy (in a Call option) or sell (in a Put	double

Identifier	Description	Type
	option) the underlying asset, and the option seller is obliged to sell or buy the appropriate amount of the underlying asset.	
SYMBOL_POINT	Symbol point value	double
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK_VALUE_PROFIT	double
SYMBOL_TRADE_TICK_VALUE_PROFIT	Calculated tick price for a profitable position	double
SYMBOL_TRADE_TICK_VALUE_LOSS	Calculated tick price for a losing position	double
SYMBOL_TRADE_TICK_SIZE	Minimal price change	double

Identifier	Description	Type
SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	double
SYMBOL_TRADE_ACCRUED_INTEREST	Accrued interest - accumulated coupon interest, i.e. part of the coupon interest calculated in proportion to the number of days since the coupon bond issuance or the last coupon interest payment	double
SYMBOL_TRADE_FACE_VALUE	Face value - initial bond value set by the issuer	double
SYMBOL_TRADE_LIQUIDITY_RATE	Liquidity Rate is the share of the	double

Identifier	Description	Type
	asset that can be used for the margin.	
SYMBOL_VOLUME_MIN	Minimal volume for a deal	double
SYMBOL_VOLUME_MAX	Maximal volume for a deal	double
SYMBOL_VOLUME_STEP	Minimal volume change step for deal execution	double
SYMBOL_VOLUME_LIMIT	Maximum allowed aggregate volume of an open position and pending orders in one direction (buy or sell) for the symbol. For example, with the limitation of 5 lots,	double

Identifier	Description	Type
	you can have an open buy position with the volume of 5 lots and place a pending order Sell Limit with the volume of 5 lots. But in this case you cannot place a Buy Limit pending order (since the total volume in one direction will exceed the limitation) or place Sell Limit with the volume more than 5 lots.	

Identifier	Description	Type
SYMBOL_SWAP_LONG	Long swap value	double
SYMBOL_SWAP_SHORT	Short swap value	double
SYMBOL_SWAP_SUNDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from SUNDAY to the next day. There following values are supported: • 0 - no swap is char	double

Identifier	Description	Type
	ged • 1 - single swap • 3 - triple swap	
SYMBOL_SWAP_MONDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from Monday to Tuesday	double

Identifier	Description	Type
SYMBOL_SWAP_TUESDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from Tuesday to Wednesday	double
SYMBOL_SWAP_WEDNESDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from Wednesday to Thursday	double
SYMBOL_SWAP_THURSDAY	Swap calculation ratio (SYMBOL_SWAP	double

Identifier	Description	Type
	_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from Thursday to Friday	
SYMBOL_SWAP_FRIDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for overnight positions rolled over from Friday to Saturday	double
SYMBOL_SWAP_SATURDAY	Swap calculation ratio (SYMBOL_SWAP_LONG or SYMBOL_SWAP_SHORT) for	double

Identifier	Description	Type
	overnight positions rolled over from Saturday to Sunday	
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening a position with the volume of one lot. It is used for checking a client's assets when he or she enters the market. The SymbolInfoMarginRate() function provides data on the amount of	double

Identifier	Description	Type
	charged margin depending on the order type and direction.	
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency of the symbol, charged from one lot. It is used for checking a client's assets when his/her account state changes. If the maintenance margin is equal to 0, the initial margin is used.	double

Identifier	Description	Type
	The SymbolInfoMarginRate() function provides data on the amount of charged margin depending on the order type and direction.	
SYMBOL_SESSION_VOLUME	Summary volume of current session deals	double
SYMBOL_SESSION_TURNOVER	Summary turnover of the current session	double
SYMBOL_SESSION_INTEREST	Summary open interest	double
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Current volume of Buy orders	double
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Current volume of Sell orders	double

Identifier	Description	Type
SYMBOL_SESSION_OPEN	Open price of the current session	double
SYMBOL_SESSION_CLOSE	Close price of the current session	double
SYMBOL_SESSION_AW	Average weighted price of the current session	double
SYMBOL_SESSION_PRICE_SETTLEMENT	Settlement price of the current session	double
SYMBOL_SESSION_PRICE_LIMIT_MIN	Minimal price of the current session	double
SYMBOL_SESSION_PRICE_LIMIT_MAX	Maximal price of the current session	double
SYMBOL_MARGIN_HEDGED	Contract size or margin value per one lot of hedged positions (oppositely directed position	double

Identifier	Description	Type
	s of one symbol). Two margin calculation methods are possible for hedged positions. The calculation method is defined by the broker. Basic calculation: • If the initial margin (SYMBOL_MARGIN_INITIAL) is specified for a symbol, the hedged margin is specified as an absol	

Identifier	Description	Type
	ute value (in monetary terms). • If the initial margin is not specified (equal to 0), SYMBOL_MARGIN_HEDGED is equal to the size of the contract, that will be used to calculate the margin by the appropriate formula in accordance with the	

Identifier	Description	Type
	type of the financial instrument (SYMBOL_TRADE_CALC_MODE). Calculation for the largest position: • The SYMBOL_MARGIN_HEDGED value is not taken into account. • The volume of all short and all long positions of a symbol is calculated. • For each	

Identifier	Description	Type
	direction, a weighted average open price and a weighted average rate of conversion to the deposit currency is calculated. - Next, using the appropriate formula chosen in accordance with the symbol type (SYMBOL_TRADE_CALC_MODE) the margi	

Identifier	Description	Type
	n is calculated for the short and the long part. • The largest one of the values is used as the margin.	
SYMBOL_PRICE_CHANGE	Change of the current price relative to the end of the previous trading day in %	double
SYMBOL_PRICE_VOLATILITY	Price volatility in %	double
SYMBOL_PRICE_THEORETICAL	Theoretical option price	double
SYMBOL_PRICE_DELTA	Option/warrant delta shows the value	double

Identifier	Description	Type
	the option price changes by, when the underlying asset price changes by 1	
SYMBOL_PRICE_THETA	Option/warrant theta shows the number of points the option price is to lose every day due to a temporary breakup, i.e. when the expiration date approaches	double
SYMBOL_PRICE_GAMMA	Option/warrant gamma shows the change rate of delta - how	double

Identifier	Description	Type
	quickly or slowly the option premium changes	
SYMBOL_PRICE_VEGA	Option/warrant vega shows the number of points the option price changes by when the volatility changes by 1%	double
SYMBOL_PRICE_RHO	Option/warrant rho reflects the sensitivity of the theoretical option price to the interest rate changing by 1%	double
SYMBOL_PRICE_OMEGA	Option/warrant omega.	double

Identifier	Description	Type
	Option elasticity shows a relative percentage change of the option price by the percentage change of the underlying asset price	
SYMBOL_PRICE_SENSITIVITY	Option/warrant sensitivity shows by how many points the price of the option's underlying asset should change so that the price of the option changes by one point	double

For function SymbolInfoString()

ENUM_SYMBOL_INFO_STRING

Identifier	Description	Type
SYMBOL_BASIS	The underlying asset of a derivative	string
SYMBOL_CATEGORY	The name of the sector or category to which the financial symbol belongs	string
SYMBOL_COUNTRY	The country to which the financial symbol belongs	string
SYMBOL_SECTOR_NAME	The sector of the economy to which the financial symbol belongs	string
SYMBOL_INDUSTRY_NAME	The industry branch or the industry to which the financial symbol belongs	string
SYMBOL_CURRENCY_BASE	Basic currency of a symbol	string

Identifier	Description	Type
SYMBOL_CURRENCY_PROFIT	Profit currency	string
SYMBOL_CURRENCY_MARGIN	Margin currency	string
SYMBOL_BANK	Feeder of the current quote	string
SYMBOL_DESCRIPTION	Symbol description	string
SYMBOL_EXCHANGE	The name of the exchange in which the financial symbol is traded	string
SYMBOL_FORMULA	The formula used for the custom symbol pricing. If the name of a financial symbol used in the formula starts with a digit or contains a special character (">" ", ".", "-", "&", "#" and so on) quotation marks	string

Identifier	Description	Type
	should be used around this symbol name. - Synthetic symbol: " @ESU19"/EURCAD - Calendarspr	

Identifier	Description	Type
	ead:"Si-9.13"-"Si-6.13" • Euroindex:34.38805726*pow	

Identifier	Description	Type
	(EURUSD,0.3155)*pow(EURGBP,0.3056)*pow(EURJPY,0.	

Identifier	Description	Type
	1891)*pow(EURCHF,0.1113)*pow(EURSEK,0.0785)	
SYMBOL_ISIN	The name of a symbol in the ISIN	string

Identifier	Description	Type
	system (International Securities Identification Number). The International Securities Identification Number is a 12-digit alphanumeric code that uniquely identifies a security. The presence of this symbol property is determined on the side of a trade server.	
SYMBOL_PAGE	The address of the web page containing symbol information. This address will be displayed as a link when viewing symbol	string

Identifier	Description	Type
	properties in the terminal	
SYMBOL_PATH	Path in the symbol tree	string

A symbol price chart can be based on Bid or Last prices. The price selected for symbol charts also affects the generation and display of bars in the terminal. Possible values of the SYMBOL_CHART_MODE property are described in ENUM_SYMBOL_CHART_MODE

ENUM_SYMBOL_CHART_MODE

Identifier	Description
SYMBOL_CHART_MODE_BID	Bars are based on Bid prices
SYMBOL_CHART_MODE_LAST	Bars are based on Last prices

For each symbol several expiration modes of pending orders can be specified. A flag is matched to each mode. Flags can be combined using the operation of logical **OR** (|), for example, SYMBOL_EXPIRATION_GTC|SYMBOL_EXPIRATION_SPECIFIED. In order to check whether a certain mode is allowed for the symbol, the result of the logical **AND** (&) should be compared to the mode flag.

If flag SYMBOL_EXPIRATION_SPECIFIED is specified for a symbol, then while sending a pending order, you may specify the moment this pending order is valid till.

Identifier	Value	Description
SYMBOL_EXPIRATION_GTC	1	The order is valid during the unlimited time period, until it is explicitly canceled
SYMBOL_EXPIRATION_DAY	2	The order is valid till the end of the day
SYMBOL_EXPIRATION_SPECIFIED	4	The expiration time is specified in the order
SYMBOL_EXPIRATION_SPECIFIED_DAY	8	The expiration date is specified in the order

Example:

```
//+------------------------------------------------------------------+
//| Checks if the specified expiration mode is allowed               |
//+------------------------------------------------------------------+
bool IsExpirationTypeAllowed(string symbol,int exp_type)
  {
//--- Obtain the value of the property that describes allowed expiration modes
   int expiration=(int)SymbolInfoInteger(symbol,SYMBOL_EXPIRATION_MODE);
//--- Return true, if mode exp_type is allowed
   return((expiration&exp_type)==exp_type);
  }
```

If the SYMBOL_EXPIRATION_MODE property is set to SYMBOL_EXPIRATION_GTC (good till canceled), the expiration of pending orders, as well as of Stop Loss/Take Profit orders should be additionally set using the ENUM_SYMBOL_ORDER_GTC_MODE enumeration.

ENUM_SYMBOL_ORDER_GTC_MODE

Identifier	Description
SYMBOL_ORDERS_GTC	Pending orders and Stop Loss/Take Profit levels are valid for an unlimited period until their explicit cancellation
SYMBOL_ORDERS_DAILY	Orders are valid during one trading day. At the end of the day, all Stop Loss and Take Profit levels, as well as pending orders are deleted.
SYMBOL_ORDERS_DAILY_EXCLUDING_STOPS	When a trade day changes, only pending orders are deleted, while Stop Loss and Take Profit levels are preserved.

When sending an order, we can specify the filling policy of a volume set in the order. The possible volume-based order execution options for each symbol are specified in the table. It is possible to set several modes for each instrument via a combination of flags. The combination of flags is expressed by the logical **OR** (|) operation, for example SYMBOL_FILLING_FOK|SYMBOL_FILLING_IOC. To check if a specific mode is allowed for an instrument, compare the logical **AND** (&) result with the mode flag - example.

Fill policy	ID	Value	Description
Fill or Kill	SYMBOL_FILLING_FOK	1	An

Fill policy	ID	Value	Description
			order can be executed in the specified volume on

Fill policy	ID	Value	Description
			ly. If the necessary amount of a financial instrum

Fill policy	ID	Value	Description
			ent is currently unavailable in the market, the or

Fill policy	ID	Value	Description
			der will not be executed. The desired volume can be

Fill policy	ID	Value	Description
			made up of several available offers. When sendi

Fill policy	ID	Value	Description
			ng an order, the ORDER_FILLING_FOK filling types

Fill policy	ID	Value	Description
			hould be specified for this policy. The possibil

Fill policy	ID	Value	Description
			ity of using FOK orders is determined at the trad

Fill policy	ID	Value	Description
			e server.
Immediate or Cancel	SYMBOL_FILLING_IOC	2	A trader agrees to execute a deal with th

Fill policy	ID	Value	Description
			e volume maximally available in the market within t

Fill policy	ID	Value	Description
			hat indicated in the order. If the request cannot

Fill policy	ID	Value	Description
			be filled completely, an order with the available vol

Fill policy	ID	Value	Description
			ume will be executed, and the remaining volume wil

Fill policy	ID	Value	Description
			l be cancelled. When sending an order, the ORDER

Fill policy	ID	Value	Description
			_FILLING_IOC filling type should be specified for

Fill policy	ID	Value	Description
			this policy. The possibility of using IOC orders i

Fill policy	ID	Value	Description
			s determined at the trade server.
Passive	SYMBOL_FILLING_BOC	4	The BOC (Book-

Fill policy	ID	Value	Description
			or-Cancel) policy assumes that an order can onl

Fill policy	ID	Value	Description
			y be placed in the Depth of Market and cannot be im

Fill policy	ID	Value	Description
			mediately executed. If the order can be execute

Fill policy	ID	Value	Description
			dimmediately when placed, then it is cancelled.

Fill policy	ID	Value	Description
			In fact, this execution policy can only be spec

Fill policy	ID	Value	Description
			ified when the price of the placed order is to be w

Fill policy	ID	Value	Description
			orse than the current market. BoC orders are use

Fill policy	ID	Value	Description
			d to implement passive trading, so that the orde

Fill policy	ID	Value	Description
			r is not executed immediately when placed and do

Fill policy	ID	Value	Description
			es not affect current liquidity. Only limit an

Fill policy	ID	Value	Description
			d stop limit orders are supported, i.e. the SYMB

Fill policy	ID	Value	Description
			OL_ORDER_MODE flag should contain the SYMBOL_

Fill policy	ID	Value	Description
			ORDER_LIMIT and/or SYMBOL_ORDER_STOP_LIMIT

Fill policy	ID	Value	Description
			values.
Return	No identifier		In case of partial filling, a market or limit o

Fill policy	ID	Value	Description
			rder with remaining volume is not canceled but pr

Fill policy	ID	Value	Description
			ocessed further. When sending an order, the OR

Fill policy	ID	Value	Description
			DER_FILLING_RETURN filling type should be specif

Fill policy	ID	Value	Description
			ied for this policy. Return orders are not allow

Fill policy	ID	Value	Description
			ed in the Market Execution mode (market executi

Fill policy	ID	Value	Description
			on_SYMBOL_TRADE_EXECUTION_MARKET).

When sending a trade request using the OrderSend() function, the necessary volume execution policy can be set in the *type_filling* field, namely in the special MqlTradeRequest structure. The values from the ENUM_ORDER_TYPE_FILLING enumeration are available. If no filling type is specified, ORDER_FILLING_RETURN is automatically set in the trade request. The ORDER_FILLING_RETURN filling type is enabled in any execution mode except for "Market execution" (SYMBOL_TRADE_EXECUTION_MARKET).

While sending a trade request for execution **at the current time** (time in force), we should keep in mind that financial markets provide no guarantee that the entire requested volume is available for a certain financial instrument at the desired price. Therefore, trading operations in real time are regulated using the price and volume execution modes. The modes, or execution policies, define the rules for cases when the price has changed or the requested volume cannot be completely fulfilled **at the moment**.

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
Execution mode (Request Execution)	Executing a market order at the price previously received from the broker. Prices for a certain market order are requested from the broker before the order is sent. Upon receiving the prices, order execution at the given price can be	SYMBOL_TRADE_EXECUTION_REQUEST

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	either confirmed or rejected.	
Instant Execution	Executing a market order at the specified price immediately. When sending a trade request to be executed, the platform automatically adds the current prices to the order. - If the broker ac	SYMBOL_TRADE_EXECUTION_INSTANT

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	cepts the price, the order is executed. • If the broker does	

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	not accept the requested price, a "Requote" is sent – the b	

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	roker returns prices, at which this order can be execute	

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	d.	
Market Execution	A broker makes a decision about the order execution price without any additional discussion with the trader. Sending the order in such a mode means advance consent to its execution at this price.	SYMBOL_TRADE_EXECUTION_MARKET
Exchange Execution	Trade operations are executed at the prices of the current	SYMBOL_TRADE_EXECUTION_EXCHANGE

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	market offers.	

Before sending an order with the current execution time, for the correct setting of the ORDER_TYPE_FILLING value (volume execution type), you can use the SymbolInfoInteger() function with each financial instrument to get the SYMBOL_FILLING_MODE property value, which shows volume execution types allowed for the symbol as a combination of flags. The ORDER_FILLING_RETURN filling type is enabled at all times except for the "Market execution" mode (SYMBOL_TRADE_EXECUTION_MARKET).

The use of filling types depending on the execution mode can be shown as the following table:

Type of Execution\Fill Policy	Fill or Kill (FOK ORDER_FILLING_FOK)	Immediate or Cancel (IOC ORDER_FILLING_IOC)	Return (Return ORDER_FILLING_RETURN)
Instant Execution (SYMBOL_TRADE_EXECUTION_INSTANT)	+ (regardless of a symbol setting)	+ (regardless of a symbol setting)	+ (always)
Request Execution SYMBOL_TRADE_EXECUTION_REQUEST	+ (regardless of a symbol setting)	+ (regardless of a symbol setting)	+ (always)
Market Execution SYMBOL_TRADE_EXECUTION_MARKET	+ (set in the symbol settings)	+ (set in the symbol settings)	- (disabled regardless of the symbol settings)
Exchange Execution SYMBOL_TRADE_EXECUTION_EXCHANGE	+ (set in the symbol settings)	+ (set in the symbol settings)	+ (always)

In case of pending orders, the ORDER_FILLING_RETURN filling type should be used regardless of an execution type (SYMBOL_TRADE_EXEMODE), since such orders are not meant for execution at the time of sending. When using pending orders, a trader agrees in advance that, when conditions for a deal on this order are met, the broker will use the filling type supported by the exchange.

Example:

```
//+------------------------------------------------------------------+
//| check if a given filling mode is allowed                         |
//+------------------------------------------------------------------+
bool IsFillingTypeAllowed(string symbol,int fill_type)
  {
//--- get the value of the property describing the filling mode
   int filling=(int)SymbolInfoInteger(symbol,SYMBOL_FILLING_MODE);
//--- return 'true' if the fill_type mode is allowed
```

```
   return((filling&fill_type)==fill_type);
  }
```

When sending a trade request using OrderSend() function, an order type from ENUM_ORDER_TYPE enumeration should be specified for some operations. Not all types of orders may be allowed for a specific symbol. SYMBOL_ORDER_MODE property describes the flags of the allowed order types.

Identifier	Value	Description
SYMBOL_ORDER_MARKET	1	Market orders are allowed (Buy and Sell)
SYMBOL_ORDER_LIMIT	2	Limit orders are allowed (Buy Limit and Sell Limit)
SYMBOL_ORDER_STOP	4	Stop orders are allowed (Buy Stop and Sell Stop)
SYMBOL_ORDER_STOP_LIMIT	8	Stop-limit orders are allowed (Buy Stop Limit and Sell Stop Limit)
SYMBOL_ORDER_SL	16	Stop Loss is allowed
SYMBOL_ORDER_TP	32	Take Profit is allowed
SYMBOL_ORDER_CLOSEBY	64	Close By operation is allowed, i.e. closing a position by another open position on the same instruments but in the opposite direction. The property is set for accounts with the hedging accounting system (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)

Example:

```
//+------------------------------------------------------------------+
//| The function prints out order types allowed for a symbol         |
//+------------------------------------------------------------------+
void Check_SYMBOL_ORDER_MODE(string symbol)
  {
//--- receive the value of the property describing allowed order types
   int symbol_order_mode=(int)SymbolInfoInteger(symbol,SYMBOL_ORDER_MODE);
//--- check for market orders (Market Execution)
   if((SYMBOL_ORDER_MARKET&symbol_order_mode)==SYMBOL_ORDER_MARKET)
      Print(symbol+": Market orders are allowed (Buy and Sell)");
//--- check for Limit orders
   if((SYMBOL_ORDER_LIMIT&symbol_order_mode)==SYMBOL_ORDER_LIMIT)
      Print(symbol+": Buy Limit and Sell Limit orders are allowed");
//--- check for Stop orders
   if((SYMBOL_ORDER_STOP&symbol_order_mode)==SYMBOL_ORDER_STOP)
      Print(symbol+": Buy Stop and Sell Stop orders are allowed");
//--- check for Stop Limit orders
   if((SYMBOL_ORDER_STOP_LIMIT&symbol_order_mode)==SYMBOL_ORDER_STOP_LIMIT)
      Print(symbol+": Buy Stop Limit and Sell Stop Limit orders are allowed");
//--- check if placing a Stop Loss orders is allowed
   if((SYMBOL_ORDER_SL&symbol_order_mode)==SYMBOL_ORDER_SL)
      Print(symbol+": Stop Loss orders are allowed");
//--- check if placing a Take Profit orders is allowed
   if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_TP)
      Print(symbol+": Take Profit orders are allowed");
//--- check if closing a position by an opposite one is allowed
   if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_CLOSEBY)
      Print(symbol+": Close by allowed");
//---
  }
```

The ENUM_SYMBOL_CALC_MODE enumeration is used for obtaining information about how the margin requirements for a symbol are calculated.

ENUM_SYMBOL_CALC_MODE

Identifier	Description	Formula
SYMBOL_CALC_MODE_FOREX	Forex mode - calculation of profit and margin for Forex	Margin: Lots * Contract_Size / Leverage * Margin_Rate Profit: (close_price - open_price) * Contract_Size*Lots
SYMBOL_CALC_MODE_FOREX_NO_LEVERAGE	Forex No Leverage mode - calculation of profit and	Margin: Lots * Contract_Size * Margin_Rate Profit: (close_price - open_price) * Contract_Size * Lots

Identifier	Description	Formula
	margin for Forex symbols without taking into account the leverage	
SYMBOL_CALC_MODE_FUTURES	Futures mode - calculation of margin and profit for futures	Margin: Lots * InitialMargin * Margin_Rate Profit: (close_price - open_price) * TickPrice / TickSize*Lots
SYMBOL_CALC_MODE_CFD	CFD mode - calculation of margin and profit for CFD	Margin: Lots * ContractSize * MarketPrice * Margin_Rate Profit: (close_price - open_price) * Contract_Size * Lots
SYMBOL_CALC_MODE_CFDINDEX	CFD inde	Margin: (Lots * ContractSize * MarketPrice) * TickPrice / TickSize * Margin_Rate

Identifier	Description	Formula
	x mode - calculation of margin and profit for CFD by indexes	Profit: (close_price - open_price) * Contract_Size * Lots
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - calculation of margin and profit for CFD at leverage trading	Margin: (Lots * ContractSize * MarketPrice) / Leverage * Margin_Rate Profit: (close_price-open_price) * Contract_Size * Lots
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode - calculation of margin and profit for	Margin: Lots * ContractSize * LastPrice * Margin_Rate Profit: (close_price - open_price) * Contract_Size * Lots

Identifier	Description	Formula
	trading securities on a stock exchange	
SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - calculation of margin and profit for trading futures contracts on a stock exchange	Margin: Lots * InitialMargin * Margin_Rate or Lots * MaintenanceMargin * Margin_Rate Profit: (close_price - open_price) * Lots * TickPrice / TickSize
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode - calculation of margin and profit for trading	Margin: Lots * InitialMargin * Margin_Rate or Lots * MaintenanceMargin * Margin_Rate Profit: (close_price - open_price) * Lots * TickPrice / TickSize

Identifier	Description	Formula
	futures contracts on FORTS. The margin may be reduced by the amount of MarginDiscount deviation according to the following rules: 1. If the price of a long position (buy order) is less than the esti	

Identifier	Description	Formula
	mated price, MarginDiscount = Lots*((PriceSettle-PriceOrder)*TickPrice/TickSize) 2. If the price of a short position (sell order) exceeds the estimated price, MarginDiscount = Lots*((PriceOrder-Price	

Identifier	Description	Formula
	Settle)*TickPrice/TickSize) where: ○ PriceSettle-estimated (clearing) price of	

Identifier	Description	Formula
	the previous session; ○ PriceOrder – average weighted po	

Identifier	Description	Formula
	sition price or open price set in the order (request); ○ Ti	

Identifier	Description	Formula
	ckPrice - tick price (cost of the price change by one poin	

Identifier	Description	Formula
	t) ○ TickSize - ticksize (minimum price change step)	
SYMBOL_CALC_MODE_EXCH_BONDS	Exchange Bonds	Margin: Lots * ContractSize * FaceValue * open_price * /100

Identifier	Description	Formula
	mode - calculation of margin and profit for trading bonds on a stock exchange	Profit: Lots * close_price * FaceValue * Contract_Size + AccruedInterest * Lots * ContractSize
SYMBOL_CALC_MODE_EXCH_STOCKS_MOEX	Exchange MOEX Stocks mode - calculation of margin and profit for trading securities on MOEX	Margin: Lots * ContractSize * LastPrice * Margin_Rate Profit: (close_price - open_price) * Contract_Size * Lots
SYMBOL_CALC_MODE_EXCH_BONDS_MOEX	Exchange MOEX Bond	Margin: Lots * ContractSize * FaceValue * open_price * /100

Identifier	Description	Formula
	s mode - calculation of margin and profit for trading bonds on MOEX	Profit: Lots * close_price * FaceValue * Contract_Size + AccruedInterest * Lots * ContractSize
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - a symbol is used as a non-tradable asset on a trading account. The market value of an open position is calculated based on	Margin: no Profit: no Market Value: Lots*ContractSize*MarketPrice*LiqudityRate

Identifier	Description	Formula
	the volume, current market price, contract size and liquidity ratio. The value is included into Assets, which are added to Equity. Open positions of such symbols increase the Free Margin in amount and are	

Identifier	Description	Formula
	used as additional margin (collateral) for open positions of tradable instruments.	

There are several symbol trading modes. Information about trading modes of a certain symbol is reflected in the values of enumeration ENUM_SYMBOL_TRADE_MODE.

ENUM_SYMBOL_TRADE_MODE

Identifier	Description
SYMBOL_TRADE_MODE_DISABLED	Trade is disabled for the symbol
SYMBOL_TRADE_MODE_LONGONLY	Allowed only long positions
SYMBOL_TRADE_MODE_SHORTONLY	Allowed only short positions
SYMBOL_TRADE_MODE_CLOSEONLY	Allowed only position close operations
SYMBOL_TRADE_MODE_FULL	No trade restrictions

Possible deal execution modes for a certain symbol are defined in enumeration ENUM_SYMBOL_TRADE_EXECUTION.

ENUM_SYMBOL_TRADE_EXECUTION

Identifier	Description
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution

Identifier	Description
SYMBOL_TRADE_EXECUTION_MARKET	Market execution
SYMBOL_TRADE_EXECUTION_EXCHANGE	Exchange execution

Methods of swap calculation at position transfer are specified in enumeration ENUM_SYMBOL_SWAP_MODE. The method of swap calculation determines the units of measure of the SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT parameters. For example, if swaps are charged in the client deposit currency, then the values of those parameters are specified as an amount of money in the client deposit currency.

ENUM_SYMBOL_SWAP_MODE

Identifier	Description
SYMBOL_SWAP_MODE_DISABLED	Swaps disabled (no swaps)
SYMBOL_SWAP_MODE_POINTS	Swaps are charged in points
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Swaps are charged in money in base currency of the symbol
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Swaps are charged in money in margin currency of the symbol
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Swaps are charged in money, in client deposit currency
SYMBOL_SWAP_MODE_CURRENCY_PROFIT	Swaps are charged in money in profit calculation currency
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Swaps are charged as the specified annual interest from the instrument price at calculation of swap (standard bank year is 360 days)
SYMBOL_SWAP_MODE_INTEREST_OPEN	Swaps are charged as the specified annual interest from the open price of position (standard bank year is 360 days)
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the close price +/- specified number of points (parameters SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)
SYMBOL_SWAP_MODE_REOPEN_BID	Swaps are charged by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the current Bid price +/- specified number of points (parameters SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)

Values of the ENUM_DAY_OF_WEEK enumeration are used for specifying days of week.

ENUM_DAY_OF_WEEK

Identifier	Description
SUNDAY	Sunday
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday

An option is a contract, which gives the right, but not the obligation, to buy or sell an underlying asset (goods, stocks, futures, etc.) at a specified price on or before a specific date. The following enumerations describe option properties, including the option type and the right arising from it.

ENUM_SYMBOL_OPTION_RIGHT

Identifier	Description
SYMBOL_OPTION_RIGHT_CALL	A call option gives you the right to buy an asset at a specified price
SYMBOL_OPTION_RIGHT_PUT	A put option gives you the right to sell an asset at a specified price

ENUM_SYMBOL_OPTION_MODE

Identifier	Description
SYMBOL_OPTION_MODE_EUROPEAN	European option may only be exercised on a specified date (expiration, execution date, delivery date)
SYMBOL_OPTION_MODE_AMERICAN	American option may be exercised on any trading day or before expiry. The period within which a buyer can exercise the option is specified for it

Financial instruments are categorized by sectors of the economy. An economic sector is a part of economic activity which has specific characteristics, economic goals, functions and behavior, which allow separating this sector from other parts of the economy. ENUM_SYMBOL_SECTOR lists the economic sectors which a trading instruments can belong to.

ENUM_SYMBOL_SECTOR

ID	Description
SECTOR_UNDEFINED	Undefined
SECTOR_BASIC_MATERIALS	Basic materials
SECTOR_COMMUNICATION_SERVICES	Communication services
SECTOR_CONSUMER_CYCLICAL	Consumer cyclical
SECTOR_CONSUMER_DEFENSIVE	Consumer defensive
SECTOR_CURRENCY	Currencies
SECTOR_CURRENCY_CRYPTO	Cryptocurrencies
SECTOR_ENERGY	Energy
SECTOR_FINANCIAL	Finance
SECTOR_HEALTHCARE	Healthcare
SECTOR_INDUSTRIALS	Industrials
SECTOR_REAL_ESTATE	Real estate
SECTOR_TECHNOLOGY	Technology
SECTOR_UTILITIES	Utilities

Each financial instrument can be assigned to a specific type of industry or economy branch. An industry is a branch of an economy that produces a closely related set of raw materials, goods, or services. ENUM_SYMBOL_INDUSTRY lists industries which a trading instrument can belong to.

ENUM_SYMBOL_INDUSTRY

ID	Description
INDUSTRY_UNDEFINED	Undefined
Basic materials	
INDUSTRY_AGRICULTURAL_INPUTS	Agricultural inputs
INDUSTRY_ALUMINIUM	Aluminium
INDUSTRY_BUILDING_MATERIALS	Building materials
INDUSTRY_CHEMICALS	Chemicals
INDUSTRY_COKING_COAL	Coking coal
INDUSTRY_COPPER	Copper
INDUSTRY_GOLD	Gold

ID	Description
INDUSTRY_LUMBER_WOOD	Lumber and wood production
INDUSTRY_INDUSTRIAL_METALS	Other industrial metals and mining
INDUSTRY_PRECIOUS_METALS	Other precious metals and mining
INDUSTRY_PAPER	Paper and paper products
INDUSTRY_SILVER	Silver
INDUSTRY_SPECIALTY_CHEMICALS	Specialty chemicals
INDUSTRY_STEEL	Steel
Communication services	
INDUSTRY_ADVERTISING	Advertising agencies
INDUSTRY_BROADCASTING	Broadcasting
INDUSTRY_GAMING_MULTIMEDIA	Electronic gaming and multimedia
INDUSTRY_ENTERTAINMENT	Entertainment
INDUSTRY_INTERNET_CONTENT	Internet content and information
INDUSTRY_PUBLISHING	Publishing
INDUSTRY_TELECOM	Telecom services
Consumer cyclical	
INDUSTRY_APPAREL_MANUFACTURING	Apparel manufacturing
INDUSTRY_APPAREL_RETAIL	Apparel retail
INDUSTRY_AUTO_MANUFACTURERS	Auto manufacturers
INDUSTRY_AUTO_PARTS	Auto parts
INDUSTRY_AUTO_DEALERSHIP	Auto and truck dealerships
INDUSTRY_DEPARTMENT_STORES	Department stores
INDUSTRY_FOOTWEAR_ACCESSORIES	Footwear and accessories
INDUSTRY_FURNISHINGS	Furnishing, fixtures and appliances
INDUSTRY_GAMBLING	Gambling
INDUSTRY_HOME_IMPROV_RETAIL	Home improvement retail
INDUSTRY_INTERNET_RETAIL	Internet retail
INDUSTRY_LEISURE	Leisure
INDUSTRY_LODGING	Lodging
INDUSTRY_LUXURY_GOODS	Luxury goods

ID	Description
INDUSTRY_PACKAGING_CONTAINERS	Packaging and containers
INDUSTRY_PERSONAL_SERVICES	Personal services
INDUSTRY_RECREATIONAL_VEHICLES	Recreational vehicles
INDUSTRY_RESIDENT_CONSTRUCTION	Residential construction
INDUSTRY_RESORTS_CASINOS	Resorts and casinos
INDUSTRY_RESTAURANTS	Restaurants
INDUSTRY_SPECIALTY_RETAIL	Specialty retail
INDUSTRY_TEXTILE_MANUFACTURING	Textile manufacturing
INDUSTRY_TRAVEL_SERVICES	Travel services
Consumer defensive	
INDUSTRY_BEVERAGES_BREWERS	Beverages - Brewers
INDUSTRY_BEVERAGES_NON_ALCO	Beverages - Non-alcoholic
INDUSTRY_BEVERAGES_WINERIES	Beverages - Wineries and distilleries
INDUSTRY_CONFECTIONERS	Confectioners
INDUSTRY_DISCOUNT_STORES	Discount stores
INDUSTRY_EDUCATION_TRAINIG	Education and training services
INDUSTRY_FARM_PRODUCTS	Farm products
INDUSTRY_FOOD_DISTRIBUTION	Food distribution
INDUSTRY_GROCERY_STORES	Grocery stores
INDUSTRY_HOUSEHOLD_PRODUCTS	Household and personal products
INDUSTRY_PACKAGED_FOODS	Packaged foods
INDUSTRY_TOBACCO	Tobacco
Energy	
INDUSTRY_OIL_GAS_DRILLING	Oil and gas drilling
INDUSTRY_OIL_GAS_EP	Oil and gas extraction and processing
INDUSTRY_OIL_GAS_EQUIPMENT	Oil and gas equipment and services
INDUSTRY_OIL_GAS_INTEGRATED	Oil and gas integrated
INDUSTRY_OIL_GAS_MIDSTREAM	Oil and gas midstream
INDUSTRY_OIL_GAS_REFINING	Oil and gas refining and marketing
INDUSTRY_THERMAL_COAL	Thermal coal

ID	Description
INDUSTRY_URANIUM	Uranium
Finance	
INDUSTRY_EXCHANGE_TRADED_FUND	Exchange traded fund
INDUSTRY_ASSETS_MANAGEMENT	Assets management
INDUSTRY_BANKS_DIVERSIFIED	Banks - Diversified
INDUSTRY_BANKS_REGIONAL	Banks - Regional
INDUSTRY_CAPITAL_MARKETS	Capital markets
INDUSTRY_CLOSE_END_FUND_DEBT	Closed-End fund - Debt
INDUSTRY_CLOSE_END_FUND_EQUITY	Closed-end fund - Equity
INDUSTRY_CLOSE_END_FUND_FOREIGN	Closed-end fund - Foreign
INDUSTRY_CREDIT_SERVICES	Credit services
INDUSTRY_FINANCIAL_CONGLOMERATE	Financial conglomerates
INDUSTRY_FINANCIAL_DATA_EXCHANGE	Financial data and stock exchange
INDUSTRY_INSURANCE_BROKERS	Insurance brokers
INDUSTRY_INSURANCE_DIVERSIFIED	Insurance - Diversified
INDUSTRY_INSURANCE_LIFE	Insurance - Life
INDUSTRY_INSURANCE_PROPERTY	Insurance - Property and casualty
INDUSTRY_INSURANCE_REINSURANCE	Insurance - Reinsurance
INDUSTRY_INSURANCE_SPECIALTY	Insurance - Specialty
INDUSTRY_MORTGAGE_FINANCE	Mortgage finance
INDUSTRY_SHELL_COMPANIES	Shell companies
Healthcare	
INDUSTRY_BIOTECHNOLOGY	Biotechnology
INDUSTRY_DIAGNOSTICS_RESEARCH	Diagnostics and research
INDUSTRY_DRUGS_MANUFACTURERS	Drugs manufacturers - general
INDUSTRY_DRUGS_MANUFACTURERS_SPEC	Drugs manufacturers - Specialty and generic
INDUSTRY_HEALTHCARE_PLANS	Healthcare plans
INDUSTRY_HEALTH_INFORMATION	Health information services
INDUSTRY_MEDICAL_FACILITIES	Medical care facilities
INDUSTRY_MEDICAL_DEVICES	Medical devices

ID	Description
INDUSTRY_MEDICAL_DISTRIBUTION	Medical distribution
INDUSTRY_MEDICAL_INSTRUMENTS	Medical instruments and supplies
INDUSTRY_PHARM_RETAILERS	Pharmaceutical retailers
Industrials	
INDUSTRY_AEROSPACE_DEFENSE	Aerospace and defense
INDUSTRY_AIRLINES	Airlines
INDUSTRY_AIRPORTS_SERVICES	Airports and air services
INDUSTRY_BUILDING_PRODUCTS	Building products and equipment
INDUSTRY_BUSINESS_EQUIPMENT	Business equipment and supplies
INDUSTRY_CONGLOMERATES	Conglomerates
INDUSTRY_CONSULTING_SERVICES	Consulting services
INDUSTRY_ELECTRICAL_EQUIPMENT	Electrical equipment and parts
INDUSTRY_ENGINEERING_CONSTRUCTION	Engineering and construction
INDUSTRY_FARM_HEAVY_MACHINERY	Farm and heavy construction machinery
INDUSTRY_INDUSTRIAL_DISTRIBUTION	Industrial distribution
INDUSTRY_INFRASTRUCTURE_OPERATIONS	Infrastructure operations
INDUSTRY_FREIGHT_LOGISTICS	Integrated freight and logistics
INDUSTRY_MARINE_SHIPPING	Marine shipping
INDUSTRY_METAL_FABRICATION	Metal fabrication
INDUSTRY_POLLUTION_CONTROL	Pollution and treatment controls
INDUSTRY_RAILROADS	Railroads
INDUSTRY_RENTAL_LEASING	Rental and leasing services
INDUSTRY_SECURITY_PROTECTION	Security and protection services
INDUSTRY_SPEALITY_BUSINESS_SERVICES	Specialty business services
INDUSTRY_SPEALITY_MACHINERY	Specialty industrial machinery
INDUSTRY_STUFFING_EMPLOYMENT	Stuffing and employment services
INDUSTRY_TOOLS_ACCESSORIES	Tools and accessories
INDUSTRY_TRUCKING	Trucking
INDUSTRY_WASTE_MANAGEMENT	Waste management
Real estate	

ID	Description
INDUSTRY_REAL_ESTATE_DEVELOPMENT	Real estate - Development
INDUSTRY_REAL_ESTATE_DIVERSIFIED	Real estate - Diversified
INDUSTRY_REAL_ESTATE_SERVICES	Real estate services
INDUSTRY_REIT_DIVERSIFIED	REIT - Diversified
INDUSTRY_REIT_HEALTCARE	REIT - Healthcase facilities
INDUSTRY_REIT_HOTEL_MOTEL	REIT - Hotel and motel
INDUSTRY_REIT_INDUSTRIAL	REIT - Industrial
INDUSTRY_REIT_MORTAGE	REIT - Mortgage
INDUSTRY_REIT_OFFICE	REIT - Office
INDUSTRY_REIT_RESIDENTAL	REIT - Residential
INDUSTRY_REIT_RETAIL	REIT - Retail
INDUSTRY_REIT_SPECIALITY	REIT - Specialty
Technology	
INDUSTRY_COMMUNICATION_EQUIPMENT	Communication equipment
INDUSTRY_COMPUTER_HARDWARE	Computer hardware
INDUSTRY_CONSUMER_ELECTRONICS	Consumer electronics
INDUSTRY_ELECTRONIC_COMPONENTS	Electronic components
INDUSTRY_ELECTRONIC_DISTRIBUTION	Electronics and computer distribution
INDUSTRY_IT_SERVICES	Information technology services
INDUSTRY_SCIENTIFIC_INSTRUMENTS	Scientific and technical instruments
INDUSTRY_SEMICONDUCTOR_EQUIPMENT	Semiconductor equipment and materials
INDUSTRY_SEMICONDUCTORS	Semiconductors
INDUSTRY_SOFTWARE_APPLICATION	Software - Application
INDUSTRY_SOFTWARE_INFRASTRUCTURE	Software - Infrastructure
INDUSTRY_SOLAR	Solar
Utilities	
INDUSTRY_UTILITIES_DIVERSIFIED	Utilities - Diversified
INDUSTRY_UTILITIES_POWERPRODUCERS	Utilities - Independent power producers
INDUSTRY_UTILITIES_RENEWABLE	Utilities - Renewable
INDUSTRY_UTILITIES_REGULATED_ELECTRIC	Utilities - Regulated electric

ID	Description
INDUSTRY_UTILITIES_REGULATED_GAS	Utilities - Regulated gas
INDUSTRY_UTILITIES_REGULATED_WATER	Utilities - Regulated water
INDUSTRY_UTILITIES_FIRST	Start of the utilities services types enumeration. Corresponds to INDUSTRY_UTILITIES_DIVERSIFIED.
INDUSTRY_UTILITIES_LAST	End of the utilities services types enumeration. Corresponds to INDUSTRY_UTILITIES_REGULATED_WATER.

Account Properties

To obtain information about the current account there are several functions: AccountInfoInteger(), AccountInfoDouble() and AccountInfoString(). The function parameter values can accept values from the corresponding ENUM_ACCOUNT_INFO enumerations.

For the function AccountInfoInteger()

ENUM_ACCOUNT_INFO_INTEGER

Identifier	Description	Type
ACCOUNT_LOGIN	Account number	long
ACCOUNT_TRADE_MODE	Account trade mode	ENUM_ACCOUNT_TRADE_MODE
ACCOUNT_LEVERAGE	Account leverage	long
ACCOUNT_LIMIT_ORDERS	Maximum allowed number of active pending orders	int
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	ENUM_ACCOUNT_STOPOUT_MODE
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	bool
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert Advisor	bool

Identifier	Description	Type
ACCOUNT_MARGIN_MODE	Margin calculation mode	ENUM_ACCOUNT_MARGIN_MODE
ACCOUNT_CURRENCY_DIGITS	The number of decimal places in the account currency, which are required for an accurate display of trading results	int
ACCOUNT_FIFO_CLOSE	An indication showing that positions can only be closed by FIFO rule. If the property value is set to true, then each symbol positions will be closed in the	bool

Identifier	Description	Type
	same order, in which they are opened, starting with the oldest one. In case of an attempt to close positions in a different order, the trader will receive an appropriate error. For accounts with the non-hedging position accounting mode (ACCOUNT_MARGIN_MODE!=ACCOUN	

Identifier	Description	Type
	T_MARGIN_MODE_RETAIL_HEDGING), the property value is always false.	
ACCOUNT_HEDGE_ALLOWED	Allowed opposite positions on a single symbol	bool

For the function AccountInfoDouble()

ENUM_ACCOUNT_INFO_DOUBLE

Identifier	Description	Type
ACCOUNT_BALANCE	Account balance in the deposit currency	double
ACCOUNT_CREDIT	Account credit in the deposit currency	double
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	double

Identifier	Description	Type
ACCOUNT_EQUITY	Account equity in the deposit currency	double
ACCOUNT_MARGIN	Account margin used in the deposit currency	double
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	double
ACCOUNT_MARGIN_LEVEL	Account margin level in percents	double
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double

Identifier	Description	Type
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double
ACCOUNT_MARGIN_INITIAL	Initial margin. The amount reserved on an account to cover the margin of all pending orders	double
ACCOUNT_MARGIN_MAINTENANCE	Maintenance margin. The minimum equity reserved on an account to cover the	double

Identifier	Description	Type
	minimum amount of all open positions	
ACCOUNT_ASSETS	The current assets of an account	double
ACCOUNT_LIABILITIES	The current liabilities on an account	double
ACCOUNT_COMMISSION_BLOCKED	The current blocked commission amount on an account	double

For function AccountInfoString()

ENUM_ACCOUNT_INFO_STRING

Identifier	Description	Type
ACCOUNT_NAME	Client name	string
ACCOUNT_SERVER	Trade server name	string
ACCOUNT_CURRENCY	Account currency	string
ACCOUNT_COMPANY	Name of a compan	string

Identifier	Description	Type
	y that serves the account	

There are several types of accounts that can be opened on a trade server. The type of account on which an MQL5 program is running can be found out using the ENUM_ACCOUNT_TRADE_MODE enumeration.

ENUM_ACCOUNT_TRADE_MODE

Identifier	Description
ACCOUNT_TRADE_MODE_DEMO	Demo account
ACCOUNT_TRADE_MODE_CONTEST	Contest account
ACCOUNT_TRADE_MODE_REAL	Real account

In case equity is not enough for maintaining open positions, the Stop Out situation, i.e. forced closing occurs. The minimum margin level at which Stop Out occurs can be set in percentage or in monetary terms. To find out the mode set for the account use the ENUM_ACCOUNT_STOPOUT_MODE enumeration.

ENUM_ACCOUNT_STOPOUT_MODE

Identifier	Description
ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money

ENUM_ACCOUNT_MARGIN_MODE

Identifier	Description
ACCOUNT_MARGIN_MODE_RETAIL_NETTING	Used for the OTC markets to interpret positions in the "netting" mode (only one position can exist for one symbol). The margin is calculated based on the symbol type (SYMBOL_TRADE_CALC_MODE).
ACCOUNT_MARGIN_MODE_EXCHANGE	Used for the exchange markets. Margin is calculated based on the discounts specified in symbol settings. Discounts are set by the broker, but not less than the values set by the exchange.
ACCOUNT_MARGIN_MODE_RETAIL_HEDGING	Used for the exchange markets where individual positions are possible (hedging, multiple positions can exist for one symbol). The margin is calculated based on the symbol

Identifier	Description
	type (SYMBOL_TRADE_CALC_MODE) taking into account the hedged margin (SYMBOL_MARGIN_HEDGED).

An example of the script that outputs a brief account information.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Name of the company
   string company=AccountInfoString(ACCOUNT_COMPANY);
//--- Name of the client
   string name=AccountInfoString(ACCOUNT_NAME);
//--- Account number
   long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- Name of the server
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- Account currency
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- Demo, contest or real account
   ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(A
//--- Now transform the value of  the enumeration into an understandable form
   string trade_mode;
   switch(account_type)
     {
      case  ACCOUNT_TRADE_MODE_DEMO:
         trade_mode="demo";
         break;
      case  ACCOUNT_TRADE_MODE_CONTEST:
         trade_mode="contest";
         break;
      default:
         trade_mode="real";
         break;
     }
//--- Stop Out is set in percentage or money
   ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInte
//--- Get the value of the levels when Margin Call and Stop Out occur
   double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
   double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- Show brief account information
   PrintFormat("The account of the client '%s' #%d %s opened in '%s' on the server '%
               name,login,trade_mode,company,server);
   PrintFormat("Account currency - %s, MarginCall and StopOut levels are set in %s",
               currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"percentage":"
   PrintFormat("MarginCall=%G, StopOut=%G",margin_call,stop_out);
  }
```

Testing Statistics

After the testing is over, different parameters of the trading results statistics are calculated. The values of the parameters can be obtained using the TesterStatistics() function, by specifying the parameter ID from the ENUM_STATISTICS enumeration.

Although two types of parameters (int and double) are used for calculating statistics, the function returns all values in the double form. All the statistic values of the double type are expressed in the deposit currency by default, unless otherwise specified.

ENUM_STATISTICS

ID	Description of a statistic parameter	Type
STAT_INITIAL_DEPOSIT	The value of the initial deposit	double
STAT_WITHDRAWAL	Money withdrawn from an account	double
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS (STAT_GROSS_LOSS is always less than or equal to zero)	double
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal to zero	double
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	double
STAT_MAX_PROFITTRADE	Maximum profit – the largest value of all profitable trades. The value is greater than or equal to zero	double
STAT_MAX_LOSSTRADE	Maximum loss – the lowest value of all losing trades. The value is less than or equal to zero	double

ID	Description of a statistic parameter	Type
STAT_CONPROFITMAX	Maximum profit in a series of profitable trades. The value is greater than or equal to zero	double
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	int
STAT_MAX_CONWINS	The total profit of the longest series of profitable trades	double
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	int
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	double
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	int
STAT_MAX_CONLOSSES	The total loss of the longest series of losing trades	double
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	int
STAT_BALANCEMIN	Minimum balance value	double
STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous	double

ID	Description of a statistic parameter	Type
	drawdowns; here the largest value is taken	
STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	double
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	double
STAT_EQUITYMIN	Minimum equity value	double
STAT_EQUITY_DD	Maximum equity drawdown in monetary terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	double
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity drawdown in monetary	double

ID	Description of a statistic parameter	Type
	terms (STAT_EQUITY_DD).	
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	double
STAT_EXPECTED_PAYOFF	Expected payoff	double
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of STAT_GROSS_PROFIT/ STAT_GROSS_LOSS. If STAT_GROSS_LOSS=0, the profit factor is equal to DBL_MAX	double
STAT_RECOVERY_FACTOR	Recovery factor, equal to the ratio of STAT_PROFIT/STAT_BALANCE_DD	double
STAT_SHARPE_RATIO	Sharpe ratio	double
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	double
STAT_CUSTOM_ONTESTER	The value of the calculated custom optimization criterion returned by the OnTester() function	double
STAT_DEALS	The number of deals	int

ID	Description of a statistic parameter	Type
STAT_TRADES	The number of trades	int
STAT_PROFIT_TRADES	Profitable trades	int
STAT_LOSS_TRADES	Losing trades	int
STAT_SHORT_TRADES	Short trades	int
STAT_LONG_TRADES	Long trades	int
STAT_PROFIT_SHORTTRADES	Profitable short trades	int
STAT_PROFIT_LONGTRADES	Profitable long trades	int
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	int
STAT_LOSSTRADES_AVGCON	Average length of a losing series of trades	int
STAT_COMPLEX_CRITERION	Complex optimization criterion	

Trade Constants

Various constants used for programming trading strategies are divided into the following groups:

- History Database Properties – receiving general information on a symbol;
- Order properties – obtaining information about trade orders;
- Position properties – obtaining information about current positions;
- Deal properties – obtaining information about deals;
- Trade operation types – description of trade operations available;
- Trade transaction types - description of possible trade transactions types;
- Trade orders in DOM – separation of orders according to the direction of a requested operation.

History Database Properties

When accessing timeseries the SeriesInfoInteger() function is used for obtaining additional symbol information. Identifier of a required property is passed as the function parameter. The identifier can be one of values of ENUM_SERIES_INFO_INTEGER.

ENUM_SERIES_INFO_INTEGER

Identifier	Description	Type
SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	long
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	datetime
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	datetime
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	datetime
SERIES_TERMINAL_FIRSTDATE	The very first date in the history of the symbol in the client terminal, regardless of the timeframe	datetime
SERIES_SYNCHRONIZED	Symbol/period data synchronization flag for the current moment	bool

Order Properties

Requests to execute trade operations are formalized as orders. Each order has a variety of properties for reading. Information on them can be obtained using functions OrderGet...() and HistoryOrderGet...().

For functions OrderGetInteger() and HistoryOrderGetInteger()

ENUM_ORDER_PROPERTY_INTEGER

Identifier	Description	Type
ORDER_TICKET	Order ticket. Unique number assigned to each order	long
ORDER_TIME_SETUP	Order setup time	datetime
ORDER_TYPE	Order type	ENUM_ORDER_TYPE
ORDER_STATE	Order state	ENUM_ORDER_STATE
ORDER_TIME_EXPIRATION	Order expiration time	datetime
ORDER_TIME_DONE	Order execution or cancellation time	datetime
ORDER_TIME_SETUP_MSC	The time of placing an order for execution in milliseconds since 01.01.1970	long
ORDER_TIME_DONE_MSC	Order execution/cancellation time in milliseconds since 01.01.1970	long
ORDER_TYPE_FILLING	Order filling type	ENUM_ORDER_TYPE_FILLING
ORDER_TYPE_TIME	Order lifetime	ENUM_ORDER_TYPE_TIME

Identifier	Description	Type
ORDER_MAGIC	ID of an Expert Advisor that has placed the order (designed to ensure that each Expert Advisor places its own unique number)	long
ORDER_REASON	The reason or source for placing an order	ENUM_ORDER_REASON
ORDER_POSITION_ID	Position identifier that is set to an order as soon as it is executed. Each executed order results in a deal that opens or modifies an already existing position. The identifier of exactly this position is set to the executed order at this moment.	long
ORDER_POSITION_BY_ID	Identifier of an opposite position used for closing by order ORDER_TYPE_CLOSE_BY	long

For functions OrderGetDouble() and HistoryOrderGetDouble()

ENUM_ORDER_PROPERTY_DOUBLE

Identifier	Description	Type
ORDER_VOLUME_INITIAL	Order initial volume	double
ORDER_VOLUME_CURRENT	Order current volume	double
ORDER_PRICE_OPEN	Price specified in the order	double
ORDER_SL	Stop Loss value	double
ORDER_TP	Take Profit value	double
ORDER_PRICE_CURRENT	The current price of the order symbol	double
ORDER_PRICE_STOPLIMIT	The Limit order price for the StopLimit order	double

For functions OrderGetString() and HistoryOrderGetString()

ENUM_ORDER_PROPERTY_STRING

Identifier	Description	Type
ORDER_SYMBOL	Symbol of the order	string
ORDER_COMMENT	Order comment	string
ORDER_EXTERNAL_ID	Order identifier in an external trading system (on the Exchange)	string

When sending a trade request using the OrderSend() function, some operations require the indication of the order type. The order type is specified in the *type* field of the special structure MqlTradeRequest, and can accept values of the ENUM_ORDER_TYPE enumeration.

ENUM_ORDER_TYPE

Identifier	Description
ORDER_TYPE_BUY	Market Buy order

Identifier	Description
ORDER_TYPE_SELL	Market Sell order
ORDER_TYPE_BUY_LIMIT	Buy Limit pending order
ORDER_TYPE_SELL_LIMIT	Sell Limit pending order
ORDER_TYPE_BUY_STOP	Buy Stop pending order
ORDER_TYPE_SELL_STOP	Sell Stop pending order
ORDER_TYPE_BUY_STOP_LIMIT	Upon reaching the order price, a pending Buy Limit order is placed at the StopLimit price
ORDER_TYPE_SELL_STOP_LIMIT	Upon reaching the order price, a pending Sell Limit order is placed at the StopLimit price
ORDER_TYPE_CLOSE_BY	Order to close a position by an opposite one

Each order has a status that describes its state. To obtain information, use OrderGetInteger() or HistoryOrderGetInteger() with the ORDER_STATE modifier. Allowed values are stored in the ENUM_ORDER_STATE enumeration.

ENUM_ORDER_STATE

Identifier	Description
ORDER_STATE_STARTED	Order checked, but not yet accepted by broker
ORDER_STATE_PLACED	Order accepted
ORDER_STATE_CANCELED	Order canceled by client
ORDER_STATE_PARTIAL	Order partially executed
ORDER_STATE_FILLED	Order fully executed
ORDER_STATE_REJECTED	Order rejected
ORDER_STATE_EXPIRED	Order expired
ORDER_STATE_REQUEST_ADD	Order is being registered (placing to the trading system)
ORDER_STATE_REQUEST_MODIFY	Order is being modified (changing its parameters)
ORDER_STATE_REQUEST_CANCEL	Order is being deleted (deleting from the trading system)

When sending a trade request for execution **at the current time** (time in force), the price and the required buy/sell volume should be specified. Also, keep in mind that financial markets provide no guarantee that the entire requested volume is available for a certain financial instrument at the desired price. Therefore, trading operations in real time are regulated using the price and volume execution modes. The modes, or execution policies, define the rules for cases when the price has changed or the requested volume cannot be completely fulfilled **at the moment**.

Price execution mode can be obtained from the SYMBOL_TRADE_EXEMODE symbol property containing the combination of flags from the ENUM_SYMBOL_TRADE_EXECUTION enumeration.

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
Execution mode (Request Execution)	Executing a market order at the price previously received from the broker. Prices for a certain market order are requested from the broker before the order is sent. Upon receiving the prices, order execution at the given price can be either confirmed or rejected.	SYMBOL_TRADE_EXECUTION_REQUEST

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
Instant Execution (Instant Execution)	Executing a market order at the specified price immediately. When sending a trade request to be executed, the platform automatically adds the current prices to the order. - If the broker accepts t	SYMBOL_TRADE_EXECUTION_INSTANT

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	he price, the order is executed. • If the broker does not acc	

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	ept the requested price, a "Requote" is sent — the broker	

Execution mode	Description	The value in ENUM_SYMBOL_TRADE_EXECUTION
	eturns prices, at which this order can be executed.	
Market Execution (Market Execution)	A broker makes a	SYMBOL_TRADE_EXECUTION_MARKET

Execution mode	Description	The value in [ENUM_SYMBOL_TRADE_EXECUTION](#)
	decision about the order execution price without any additional discussion with the trader. Sending the order in such a mode means advance consent to its execution at this price.	
Exchange Execution (Exchange Execution)	Trade operations are executed at the prices of the current market offers.	SYMBOL_TRADE_EXECUTION_EXCHANGE

Volume filling policy is specified in the ORDER_TYPE_FILLING order property and may contain only the values from the ENUM_ORDER_TYPE_FILLING enumeration

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
Fill or Kill	An order can be executed in the specified volume only. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The desired volume can be made up of several available offers. The possibi	ORDER_FILLING_FOK

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
	lity of using FOK orders is determined at the trade server.	
Immediate or Cancel	A trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. If the request cannot be filled completely, an order with the available volume	ORDER_FILLING_IOC

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
	will be executed, and the remaining volume will be canceled. The possibility of using IOC orders is determined at the trade server.	
Passive (Book or Cancel)	The BoC order assumes that the order can only be placed in the Depth of Market and cannot be immediately executed. If the order	ORDER_FILLING_BOC

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
	can be executed immediately when placed, then it is canceled. In fact, the BOC policy guarantees that the price of the placed order will be worse than the current market. BoC orders are used to implement passive trading, so that the order is not executed	

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
	immediately when placed and does not affect current liquidity. Only limit and stop limit orders are supported (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT, ORDER_TYPE_SELL_STOP_LIMIT).	
Return	In case of partial filling, an order with	ORDER_FILLING_RETURN

Fill policy	Description	The value in ENUM_ORDER_TYPE_FILLING
	remaining volume is not canceled but processed further. Return orders are not allowed in the Market Execution mode (market execution — SYMBOL_TRADE_EXECUTION_MARKET).	

When sending a trade request using the OrderSend() function, the necessary volume execution policy can be set in the *type_filling* field, namely in the special MqlTradeRequest structure. The values from the ENUM_ORDER_TYPE_FILLING enumeration are available. To get the property value in a specific active/completed order, use the OrderGetInteger() or HistoryOrderGetInteger() function with the ORDER_TYPE_FILLING modifier.

Before sending an order with the current execution time, for the correct setting of the ORDER_TYPE_FILLING value (volume execution type), you can use the SymbolInfoInteger() function with each financial instrument to get the SYMBOL_FILLING_MODE property value, which shows volume execution types allowed for the symbol as a combination of flags. The ORDER_FILLING_RETURN filling type is enabled at all times except for the "Market execution" mode (SYMBOL_TRADE_EXECUTION_MARKET).

The use of filling types depending on the execution mode can be shown as the following table:

Type of Execution\Fill Policy	Fill or Kill (FOK ORDER_FILLING_FOK)	Immediate or Cancel (IOC ORDER_FILLING_IOC)	Return (Return ORDER_FILLING_RETURN)
Instant Execution (SYMBOL_TRADE_EXECUTION_INSTANT)	+ (regardless of a symbol setting)	+ (regardless of a symbol setting)	+ (always)
Request Execution SYMBOL_TRADE_EXECUTION_REQUEST	+ (regardless of a symbol setting)	+ (regardless of a symbol setting)	+ (always)
Market Execution SYMBOL_TRADE_EXECUTION_MARKET	+ (set in the symbol settings)	+ (set in the symbol settings)	- (disabled regardless of the symbol settings)
Exchange Execution SYMBOL_TRADE_EXECUTION_EXCHANGE	+ (set in the symbol settings)	+ (set in the symbol settings)	+ (always)

In case of pending orders, the ORDER_FILLING_RETURN filling type should be used regardless of an execution type (SYMBOL_TRADE_EXEMODE), since such orders are not meant for execution at the time of sending. When using pending orders, a trader agrees in advance that, when conditions for a deal on this order are met, the broker will use the filling type supported by the exchange.

The order validity period can be set in the *type_time* field of the special structure MqlTradeRequest when sending a trade request using the OrderSend() function. Values of the ENUM_ORDER_TYPE_TIME enumeration are allowed. To obtain the value of this property use the function OrderGetInteger() or HistoryOrderGetInteger() with the ORDER_TYPE_TIME modifier.

ENUM_ORDER_TYPE_TIME

Identifier	Description
ORDER_TIME_GTC	Good till cancel order
ORDER_TIME_DAY	Good till current trade day order
ORDER_TIME_SPECIFIED	Good till expired order
ORDER_TIME_SPECIFIED_DAY	The order will be effective till 23:59:59 of the specified day. If this time is outside a trading session, the order expires in the nearest trading time.

The reason for order placing is contained in the ORDER_REASON property. An order can be placed by an MQL5 program, from a mobile application, as a result of StopOut, etc. Possible values of ORDER_REASON are described in the ENUM_ORDER_REASON enumeration.

ENUM_ORDER_REASON

Identifier	Description
ORDER_REASON_CLIENT	The order was placed from a desktop terminal
ORDER_REASON_MOBILE	The order was placed from a mobile application
ORDER_REASON_WEB	The order was placed from a web platform
ORDER_REASON_EXPERT	The order was placed from an MQL5-program, i.e. by an Expert Advisor or a script
ORDER_REASON_SL	The order was placed as a result of Stop Loss activation
ORDER_REASON_TP	The order was placed as a result of Take Profit activation
ORDER_REASON_SO	The order was placed as a result of the Stop Out event

Position Properties

Execution of trade operations results in the opening of a position, changing of its volume and/or direction, or its disappearance. Trade operations are conducted based on orders, sent by the OrderSend() function in the form of trade requests. For each financial security (symbol) only one open position is possible. A position has a set of properties available for reading by the PositionGet...() functions.

For the function PositionGetInteger()

ENUM_POSITION_PROPERTY_INTEGER

Identifier	Description	Type
POSITION_TICKET	Position ticket. Unique number assigned to each newly opened position. It usually matches the ticket of an order used to open the position except when the ticket is changed as a result of service operations on the server, for example, when charging swaps with position re-opening. To find an order used to open a position, apply the POSITION_IDENTIFIER property. POSITION_TICKET value corresponds to MqlTradeRequest::position.	long
POSITION_TIME	Position open time	datetime
POSITION_TIME_MSC	Position opening time in milliseconds since 01.01.1970	long

Identifier	Description	Type
POSITION_TIME_UPDATE	Position changing time	datetime
POSITION_TIME_UPDATE_MSC	Position changing time in milliseconds since 01.01.1970	long
POSITION_TYPE	Position type	ENUM_POSITION_TYPE
POSITION_MAGIC	Position magic number (see ORDER_MAGIC)	long
POSITION_IDENTIFIER	Position identifier is a unique number assigned to each re-opened position. It does not change throughout its life cycle and corresponds to the ticket of an order used to open a position. Position identifier is specified in each order (ORDER_POSITION_ID) and deal (DEAL_POSITION_ID) used to open, modify, or close it. Use this property to search for orders and deals related to the position. When reversing a position in netting mode (using a single in/out trade), POSITION_IDENTIFIER does not change. However, POSITION_TICKET is replaced with	long

Identifier	Description	Type
	the ticket of the order that led to the reversal. Position reversal is not provided in hedging mode.	
POSITION_REASON	The reason for opening a position	ENUM_POSITION_REASON

For the function PositionGetDouble()

ENUM_POSITION_PROPERTY_DOUBLE

Identifier	Description	Type
POSITION_VOLUME	Position volume	double
POSITION_PRICE_OPEN	Position open price	double
POSITION_SL	Stop Loss level of opened position	double
POSITION_TP	Take Profit level of opened position	double
POSITION_PRICE_CURRENT	Current price of the position symbol	double
POSITION_SWAP	Cumulative swap	double
POSITION_PROFIT	Current profit	double

For the function PositionGetString()

ENUM_POSITION_PROPERTY_STRING

Identifier	Description	Type
POSITION_SYMBOL	Symbol of the position	string
POSITION_COMMENT	Position comment	string
POSITION_EXTERNAL_ID	Position identifier in an external trading system (on the Exchange)	string

Direction of an open position (buy or sell) is defined by the value from the ENUM_POSITION_TYPE enumeration. In order to obtain the type of an open position use the PositionGetInteger() function with the POSITION_TYPE modifier.

ENUM_POSITION_TYPE

Identifier	Description
POSITION_TYPE_BUY	Buy
POSITION_TYPE_SELL	Sell

The reason for opening a position is contained in the POSITION_REASON property. A position can be opened as a result of activation of an order placed from a desktop terminal, a mobile application, by an Expert Advisor, etc. Possible values of POSITION_REASON are described in the ENUM_POSITION_REASON enumeration.

ENUM_POSITION_REASON

Identifier	Description
POSITION_REASON_CLIENT	The position was opened as a result of activation of an order placed from a desktop terminal
POSITION_REASON_MOBILE	The position was opened as a result of activation of an order placed from a mobile application
POSITION_REASON_WEB	The position was opened as a result of activation of an order placed from the web platform
POSITION_REASON_EXPERT	The position was opened as a result of activation of an order placed from an MQL5 program, i.e. an Expert Advisor or a script

Deal Properties

A deal is the reflection of the fact of a trade operation execution based on an order that contains a trade request. Each trade is described by properties that allow to obtain information about it. In order to read values of properties, functions of the HistoryDealGet...() type are used, that return values from corresponding enumerations.

For the function HistoryDealGetInteger()

ENUM_DEAL_PROPERTY_INTEGER

Identifier	Description	Type
DEAL_TICKET	Deal ticket. Unique number assigned to each deal	long
DEAL_ORDER	Deal order number	long
DEAL_TIME	Deal time	datetime
DEAL_TIME_MSC	The time of a deal execution in milliseconds since 01.01.1970	long
DEAL_TYPE	Deal type	ENUM_DEAL_TYPE
DEAL_ENTRY	Deal entry - entry in, entry out, reverse	ENUM_DEAL_ENTRY
DEAL_MAGIC	Deal magic number (see ORDER_MAGIC)	long
DEAL_REASON	The reason or source for deal execution	ENUM_DEAL_REASON
DEAL_POSITION_ID	Identifier of a position, in the opening, modification or closing of which this deal took part. Each position has a unique identifier that is assigned to all deals executed for the symbol during the entire lifetime of the position.	long

For the function HistoryDealGetDouble()

ENUM_DEAL_PROPERTY_DOUBLE

Identifier	Description	Type
DEAL_VOLUME	Deal volume	double
DEAL_PRICE	Deal price	double
DEAL_COMMISSION	Deal commission	double
DEAL_SWAP	Cumulative swap on close	double
DEAL_PROFIT	Deal profit	double
DEAL_FEE	Fee for making a deal charged immediately after performing a deal	double

Identifier	Description	Type
DEAL_SL	Stop Loss level • Entry and reversal deals use the Stop Loss values from the original order based on which the position was opened or reversed • Exit deals use the Stop Loss of a position as at the time of position closing	double
DEAL_TP	Take Profit level • Entry and reversal deals use the Take Profit values from the original order based on which the position was opened or reversed • Exit deals use the Take Profit value of a position as at the time of position closing	double

For the function HistoryDealGetString()

ENUM_DEAL_PROPERTY_STRING

Identifier	Description	Type
DEAL_SYMBOL	Deal symbol	string
DEAL_COMMENT	Deal comment	string
DEAL_EXTERNAL_ID	Deal identifier in an external trading system (on the Exchange)	string

Each deal is characterized by a type, allowed values are enumerated in ENUM_DEAL_TYPE. In order to obtain information about the deal type, use the HistoryDealGetInteger() function with the DEAL_TYPE modifier.

ENUM_DEAL_TYPE

Identifier	Description
DEAL_TYPE_BUY	Buy
DEAL_TYPE_SELL	Sell
DEAL_TYPE_BALANCE	Balance
DEAL_TYPE_CREDIT	Credit

Identifier	Description
DEAL_TYPE_CHARGE	Additional charge
DEAL_TYPE_CORRECTION	Correction
DEAL_TYPE_BONUS	Bonus
DEAL_TYPE_COMMISSION	Additional commission
DEAL_TYPE_COMMISSION_DAILY	Daily commission
DEAL_TYPE_COMMISSION_MONTHLY	Monthly commission
DEAL_TYPE_COMMISSION_AGENT_DAILY	Daily agent commission
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Monthly agent commission
DEAL_TYPE_INTEREST	Interest rate
DEAL_TYPE_BUY_CANCELED	Canceled buy deal. There can be a situation when a previously executed buy deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_BUY) is changed to DEAL_TYPE_BUY_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation
DEAL_TYPE_SELL_CANCELED	Canceled sell deal. There can be a situation when a previously executed sell deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_SELL) is changed to DEAL_TYPE_SELL_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn using a separated balance operation
DEAL_DIVIDEND	Dividend operations
DEAL_DIVIDEND_FRANKED	Franked (non-taxable) dividend operations
DEAL_TAX	Tax charges

Deals differ not only in their types set in ENUM_DEAL_TYPE, but also in the way they change positions. This can be a simple position opening, or accumulation of a previously opened position (market entering), position closing by an opposite deal of a corresponding volume (market exiting), or position reversing, if the opposite-direction deal covers the volume of the previously opened position.

All these situations are described by values from the ENUM_DEAL_ENTRY enumeration. In order to receive this information about a deal, use the HistoryDealGetInteger() function with the DEAL_ENTRY modifier.

ENUM_DEAL_ENTRY

Identifier	Description
DEAL_ENTRY_IN	Entry in
DEAL_ENTRY_OUT	Entry out
DEAL_ENTRY_INOUT	Reverse
DEAL_ENTRY_OUT_BY	Close a position by an opposite one

The reason for deal execution is contained in the DEAL_REASON property. A deal can be executed as a result of triggering of an order placed from a mobile application or an MQL5 program, as well as as a result of the StopOut event, variation margin calculation, etc. Possible values of DEAL_REASON are described in the ENUM_DEAL_REASON enumeration. For non-trading deals resulting from balance, credit, commission and other operations, DEAL_REASON_CLIENT is indicated as the reason.

ENUM_DEAL_REASON

Identifier	Description
DEAL_REASON_CLIENT	The deal was executed as a result of activation of an order placed from a desktop terminal
DEAL_REASON_MOBILE	The deal was executed as a result of activation of an order placed from a mobile application
DEAL_REASON_WEB	The deal was executed as a result of activation of an order placed from the web platform
DEAL_REASON_EXPERT	The deal was executed as a result of activation of an order placed from an MQL5 program, i.e. an Expert Advisor or a script
DEAL_REASON_SL	The deal was executed as a result of Stop Loss activation
DEAL_REASON_TP	The deal was executed as a result of Take Profit activation
DEAL_REASON_SO	The deal was executed as a result of the Stop Out event
DEAL_REASON_ROLLOVER	The deal was executed due to a rollover
DEAL_REASON_VMARGIN	The deal was executed after charging the variation margin
DEAL_REASON_SPLIT	The deal was executed after the split (price reduction) of an instrument, which had an open position during split announcement
DEAL_REASON_CORPORATE_ACTION	The deal was executed as a result of a corporate action: merging or renaming a security, transferring a client to another account, etc.

Trade Operation Types

Trading is done by sending orders to open positions using the OrderSend() function, as well as to place, modify or delete pending orders. Each trade order refers to the type of the requested operation. Trading operations are described in the ENUM_TRADE_REQUEST_ACTIONS enumeration.

ENUM_TRADE_REQUEST_ACTIONS

Identifier	Description
TRADE_ACTION_DEAL	Place a trade order for an immediate execution with the specified parameters (market order)
TRADE_ACTION_PENDING	Place a trade order for the execution under specified conditions (pending order)
TRADE_ACTION_SLTP	Modify Stop Loss and Take Profit values of an opened position
TRADE_ACTION_MODIFY	Modify the parameters of the order placed previously
TRADE_ACTION_REMOVE	Delete the pending order placed previously
TRADE_ACTION_CLOSE_BY	Close a position by an opposite one

Example of the TRADE_ACTION_DEAL trade operation for opening a Buy position:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Opening Buy position                                             |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters of request
   request.action   =TRADE_ACTION_DEAL;                     // type of trade operatio
   request.symbol   =Symbol();                              // symbol
   request.volume   =0.1;                                   // volume of 0.1 lot
   request.type     =ORDER_TYPE_BUY;                        // order type
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // price for opening
   request.deviation=5;                                     // allowed deviation from
   request.magic    =EXPERT_MAGIC;                          // MagicNumber of the orde
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // if unable to send the
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_DEAL trade operation for opening a Sell position:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Opening Sell position                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters of request
   request.action   =TRADE_ACTION_DEAL;                      // type of trade operatio
   request.symbol   =Symbol();                               // symbol
   request.volume   =0.2;                                    // volume of 0.2 lot
   request.type     =ORDER_TYPE_SELL;                        // order type
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_BID); // price for opening
   request.deviation=5;                                     // allowed deviation from
   request.magic    =EXPERT_MAGIC;                          // MagicNumber of the orde
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // if unable to send the
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_DEAL trade operation for closing positions:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Closing all positions                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
   for(int i=total-1; i>=0; i--)
     {
      //--- parameters of the order
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- output information about the position
      PrintFormat("#%I64u %s  %s  %.2f  %s [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  magic);
      //--- if the MagicNumber matches
      if(magic==EXPERT_MAGIC)
        {
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action   =TRADE_ACTION_DEAL;        // type of trade operation
         request.position =position_ticket;          // ticket of the position
         request.symbol   =position_symbol;          // symbol
         request.volume   =volume;                   // volume of the position
         request.deviation=5;                        // allowed deviation from the pri
         request.magic    =EXPERT_MAGIC;             // MagicNumber of the position
         //--- set the price and order type depending on the position type
         if(type==POSITION_TYPE_BUY)
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
            request.type =ORDER_TYPE_SELL;
           }
         else
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
            request.type =ORDER_TYPE_BUY;
           }
         //--- output information about the closure
         PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToStrin
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //---
        }
        }
   }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_PENDING trade operation for placing a pending order:

```
#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456                             // MagicNumber of the expert
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;   // order type
//+------------------------------------------------------------------+
//| Placing pending orders                                           |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters to place a pending order
   request.action   =TRADE_ACTION_PENDING;                             // type of trac
   request.symbol   =Symbol();                                         // symbol
   request.volume   =0.1;                                              // volume of 0
   request.deviation=2;                                                // allowed dev
   request.magic    =EXPERT_MAGIC;                                     // MagicNumber
   int offset = 50;                                                    // offset from
   double price;                                                       // order trigge
   double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT);                // value of po
   int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);                // number of de
   //--- checking the type of operation
   if(orderType==ORDER_TYPE_BUY_LIMIT)
     {
      request.type     =ORDER_TYPE_BUY_LIMIT;                          // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;        // price for op
      request.price    =NormalizeDouble(price,digits);                 // normalized
     }
   else if(orderType==ORDER_TYPE_SELL_LIMIT)
     {
      request.type     =ORDER_TYPE_SELL_LIMIT;                          // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;         // price for
      request.price    =NormalizeDouble(price,digits);                  // normalized
     }
   else if(orderType==ORDER_TYPE_BUY_STOP)
     {
      request.type =ORDER_TYPE_BUY_STOP;                                // order type
      price        =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for
      request.price=NormalizeDouble(price,digits);                      // normalized
     }
   else if(orderType==ORDER_TYPE_SELL_STOP)
     {
      request.type     =ORDER_TYPE_SELL_STOP;                           // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;         // price for
      request.price    =NormalizeDouble(price,digits);                  // normalized
     }
   else Alert("This example is only for placing pending orders");   // if not pending
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());                 // if unable
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_SLTP trade operation for modifying the Stop Loss and Take Profit values of an open position:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Modification of Stop Loss and Take Profit of position            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
   for(int i=0; i<total; i++)
     {
      //--- parameters of the order
      ulong  position_ticket=PositionGetTicket(i);// ticket of the position
      string position_symbol=PositionGetString(POSITION_SYMBOL); // symbol
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // number
      ulong  magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber of the position
      double volume=PositionGetDouble(POSITION_VOLUME);    // volume of the position
      double sl=PositionGetDouble(POSITION_SL);  // Stop Loss of the position
      double tp=PositionGetDouble(POSITION_TP);  // Take Profit of the position
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- output information about the position
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         //--- calculate the current price levels
         double price=PositionGetDouble(POSITION_PRICE_OPEN);
         double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
         double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
         int    stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_
         double price_level;
         //--- if the minimum allowed offset distance in points from the current clos
         if(stop_level<=0)
            stop_level=150; // set the offset distance of 150 points from the current
         else
            stop_level+=50; // set the offset distance to (SYMBOL_TRADE_STOPS_LEVEL +

         //--- calculation and rounding of the Stop Loss and Take Profit values
         price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
         if(type==POSITION_TYPE_BUY)
           {
            sl=NormalizeDouble(bid-price_level,digits);
            tp=NormalizeDouble(ask+price_level,digits);
           }
         else
           {
            sl=NormalizeDouble(ask+price_level,digits);
            tp=NormalizeDouble(bid-price_level,digits);
           }
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action  =TRADE_ACTION_SLTP; // type of trade operation
         request.position=position_ticket;   // ticket of the position
         request.symbol=position_symbol;     // symbol
         request.sl      =sl;                // Stop Loss of the position
         request.tp      =tp;                // Take Profit of the position
         request.magic=EXPERT_MAGIC;         // MagicNumber of the position
         //--- output information about the modification
         PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToStrin
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_MODIFY trade operation for modifying the price levels of pending orders:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Modification of pending orders                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
   for(int i=0; i<total; i++)
     {
      //--- parameters of the order
      ulong  order_ticket=OrderGetTicket(i);                             // order tic
      string order_symbol=Symbol();                                      // symbol
      int    digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS);  // number of
      ulong  magic=OrderGetInteger(ORDER_MAGIC);                         // MagicNumb
      double volume=OrderGetDouble(ORDER_VOLUME_CURRENT);                // current v
      double sl=OrderGetDouble(ORDER_SL);                                // current S
      double tp=OrderGetDouble(ORDER_TP);                                // current T
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // type of t
      int offset = 50;                                                   // offset fr
      double price;                                                      // order tri
      double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT);          // value of
      //--- output information about the order
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  order_ticket,
                  order_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         request.action=TRADE_ACTION_MODIFY;                           // type of tra
         request.order = OrderGetTicket(i);                            // order ticke
         request.symbol   =Symbol();                                   // symbol
         request.deviation=5;                                          // allowed dev
        //--- setting the price level, Take Profit and Stop Loss of the order dependin
         if(type==ORDER_TYPE_BUY_LIMIT)
           {
            price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                // normal
           }
         else if(type==ORDER_TYPE_SELL_LIMIT)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                 // norma
           }
         else if(type==ORDER_TYPE_BUY_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                // normal
           }
         else if(type==ORDER_TYPE_SELL_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                // normal
           }
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
        }
     }
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_REMOVE trade operation for deleting pending orders:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Deleting pending orders                                          |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
   for(int i=total-1; i>=0; i--)
     {
      ulong  order_ticket=OrderGetTicket(i);                  // order ticket
      ulong  magic=OrderGetInteger(ORDER_MAGIC);              // MagicNumber of the o
      //--- if the MagicNumber matches
      if(magic==EXPERT_MAGIC)
        {
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action=TRADE_ACTION_REMOVE;                  // type of trade operat
         request.order = order_ticket;                        // order ticket
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_CLOSE_BY trade operation for closing positions by opposite positions:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Close all positions by opposite positions                        |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
   for(int i=total-1; i>=0; i--)
     {
      //--- parameters of the order
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      double sl=PositionGetDouble(POSITION_SL);
      double tp=PositionGetDouble(POSITION_TP);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- output information about the position
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- if the MagicNumber matches
      if(magic==EXPERT_MAGIC)
        {
         for(int j=0; j<i; j++)
           {
            string symbol=PositionGetSymbol(j); // symbol of the opposite position
            //--- if the symbols of the opposite and initial positions match
            if(symbol==position_symbol && PositionGetInteger(POSITION_MAGIC)==EXPERT_M
              {
               //--- set the type of the opposite position
               ENUM_POSITION_TYPE type_by=(ENUM_POSITION_TYPE)PositionGetInteger(POSIT
               //--- leave, if the types of the initial and opposite positions match
               if(type==type_by)
                  continue;
               //--- zeroing the request and result values
               ZeroMemory(request);
               ZeroMemory(result);
               //--- setting the operation parameters
               request.action=TRADE_ACTION_CLOSE_BY;                         // type o
               request.position=position_ticket;                             // ticket
               request.position_by=PositionGetInteger(POSITION_TICKET);      // ticket
               //request.symbol     =position_symbol;
               request.magic=EXPERT_MAGIC;                                   // MagicN
               //--- output information about the closure by opposite position
               PrintFormat("Close #%I64d %s %s by #%I64d",position_ticket,position_sym
               //--- send the request
               if(!OrderSend(request,result))
                  PrintFormat("OrderSend error %d",GetLastError()); // if unable to se
```

```
                 //--- information about the operation
                 PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result
                }
             }
          }
       }
    }
//+------------------------------------------------------------------+
```

Trade Transaction Types

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using OrderSend and OrderSendAsync functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions.

To let a programmer to track the actions performed in relation to a trade account, OnTradeTransaction function has been provided. This handler allows to get trade transactions applied to an account in MQL5 application. Trade transaction description is submitted in OnTradeTransaction first parameter using MqlTradeTransaction structure.

Trade transaction type is submitted in the type parameter of MqlTradeTransaction structure. Possible types of trade transactions are described by the following enumeration:

ENUM_TRADE_TRANSACTION_TYPE

Identifier	Description
TRADE_TRANSACTION_ORDER_ADD	Adding a new open order.
TRADE_TRANSACTION_ORDER_UPDATE	Updating an open order. The updates include not only evident changes from the client terminal or a trade server sides but also changes of an order state when setting it (for example, transition from ORDER_STATE_STARTED to ORDER_STATE_PLACED or from ORDER_STATE_PLACED to ORDER_STATE_PARTIAL, etc.).
TRADE_TRANSACTION_ORDER_DELETE	Removing an order from the list of the open ones. An order can be deleted from the open ones as a result of setting an appropriate request or execution (filling) and moving to the history.
TRADE_TRANSACTION_DEAL_ADD	Adding a deal to the history. The action is performed as a result of an order execution or performing

Identifier	Description
	operations with an account balance.
TRADE_TRANSACTION_DEAL_UPDATE	Updating a deal in the history. There may be cases when a previously executed deal is changed on a server. For example, a deal has been changed in an external trading system (exchange) where it was previously transferred by a broker.
TRADE_TRANSACTION_DEAL_DELETE	Deleting a deal from the history. There may be cases when a previously executed deal is deleted from a server. For example, a deal has been deleted in an external trading system (exchange) where it was previously transferred by a broker.
TRADE_TRANSACTION_HISTORY_ADD	Adding an order to the history as a result of execution or cancellation.
TRADE_TRANSACTION_HISTORY_UPDATE	Changing an order located in the orders history. This type is provided for enhancing functionality on a trade server side.
TRADE_TRANSACTION_HISTORY_DELETE	Deleting an order from the orders history. This type is provided for enhancing functionality on a trade server side.
TRADE_TRANSACTION_POSITION	Changing a position not related to a deal execution. This type of transaction shows that a position has been changed on a trade server side. Position volume, open price, Stop Loss and Take Profit levels can be changed. Data on changes are submitted in MqlTradeTransaction structure via OnTradeTransaction handler. Position change (adding, changing or closing), as a result of a deal execution, does not lead to the occurrence of TRADE_TRANSACTION_POSITION transaction.
TRADE_TRANSACTION_REQUEST	Notification of the fact that a trade request has been processed by a server and processing result has been received. Only type field (trade transaction type) must be analyzed for such transactions in MqlTradeTransaction structure. The second and third parameters of OnTradeTransaction (request and result) must be analyzed for additional data.

Depending on a trade transaction type, various parameters are filled in MqlTradeTransaction structure describing it. A detailed description of submitted data is shown in "Structure of a Trade Transaction".

See also

Structure of a Trade Transaction, OnTradeTransaction

Trade Orders in Depth Of Market

For equity securities, the Depth of Market window is available, where you can see the current Buy and Sell orders. Desired direction of a trade operation, required amount and requested price are specified for each order.

To obtain information about the current state of the DOM by MQL5 means, the MarketBookGet() function is used, which places the DOM "screen shot" into the MqlBookInfo array of structures. Each element of the array in the *type* field contains information about the direction of the order - the value of the ENUM_BOOK_TYPE enumeration.

ENUM_BOOK_TYPE

Identifier	Description
BOOK_TYPE_SELL	Sell order (Offer)
BOOK_TYPE_BUY	Buy order (Bid)
BOOK_TYPE_SELL_MARKET	Sell order by Market
BOOK_TYPE_BUY_MARKET	Buy order by Market

See also

Structures and classes, Structure of the DOM, Trade operation types, Market Info

Signal Properties

The following enumerations are used when working with trading signals and signal copy settings.

Enumeration of double type properties of the trading signal:

ENUM_SIGNAL_BASE_DOUBLE

ID	Description
SIGNAL_BASE_BALANCE	Account balance
SIGNAL_BASE_EQUITY	Account equity
SIGNAL_BASE_GAIN	Account gain
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown
SIGNAL_BASE_PRICE	Signal subscription price
SIGNAL_BASE_ROI	Return on Investment (%)

Enumeration of integer type properties of the trading signal:

ENUM_SIGNAL_BASE_INTEGER

ID	Description
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)
SIGNAL_BASE_DATE_STARTED	Monitoring starting date
SIGNAL_BASE_DATE_UPDATED	The date of the last update of the signal's trading statistics
SIGNAL_BASE_ID	Signal ID
SIGNAL_BASE_LEVERAGE	Account leverage
SIGNAL_BASE_PIPS	Profit in pips
SIGNAL_BASE_RATING	Position in rating
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers
SIGNAL_BASE_TRADES	Number of trades
SIGNAL_BASE_TRADE_MODE	Account type (0-real, 1-demo, 2-contest)

Enumeration of string type properties of the trading signal:

ENUM_SIGNAL_BASE_STRING

ID	Description
SIGNAL_BASE_AUTHOR_LOGIN	Author login
SIGNAL_BASE_BROKER	Broker name (company)

ID	Description
SIGNAL_BASE_BROKER_SERVER	Broker server
SIGNAL_BASE_NAME	Signal name
SIGNAL_BASE_CURRENCY	Signal base currency

Enumeration of double type properties of the signal copy settings:

ENUM_SIGNAL_INFO_DOUBLE

ID	Description
SIGNAL_INFO_EQUITY_LIMIT	Equity limit
SIGNAL_INFO_SLIPPAGE	Slippage (used when placing market orders in synchronization of positions and copying of trades)
SIGNAL_INFO_VOLUME_PERCENT	Maximum percent of deposit used (%), r/o

Enumeration of integer type properties of the signal copy settings:

ENUM_SIGNAL_INFO_INTEGER

ID	Description
SIGNAL_INFO_CONFIRMATIONS_DISABLED	The flag enables synchronization without confirmation dialog
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag
SIGNAL_INFO_DEPOSIT_PERCENT	Deposit percent (%)
SIGNAL_INFO_ID	Signal id, r/o
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o

Enumeration of string type properties of the signal copy settings:

ENUM_SIGNAL_INFO_STRING

ID	Description
SIGNAL_INFO_NAME	Signal name, r/o

See also

Trade signals

Named Constants

All constants used in MQL5 can be divided into the following groups:

- Predefined macro substitutions – values are substituted during compilation;
- Mathematical constants – values of some mathematical expressions;
- Numerical type constants – some of the simple type restrictions;
- Uninitialization reason codes – description of uninitialization reasons;
- Checking Object Pointer – enumeration of types of pointers returned by the CheckPointer() function;
- Other constants – all other constants.

Predefined Macro Substitutions

To simplify the debugging process and obtain information about operation of a mql5-program, there are special macro constant, values of which are set at the moment of compilation. The easiest way to use these constants is outputting values by the Print() function, as it's shown in the example.

Constant	Description
__CPU_ARCHITECTURE__	Name of architecture (set of commands) EX5 file compiled for
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)
__DATETIME__	File compilation date and time
__LINE__	Line number in the source code, in which the macro is located
__FILE__	Name of the currently compiled file
__PATH__	An absolute path to the file that is currently being compiled
__FUNCTION__	Name of the function, in whose body the macro is located
__FUNCSIG__	Signature of the function in whose body the macro is located. Logging of the full description of functions can be useful in the identification of overloaded functions
__MQLBUILD__,__MQL5BUILD__	Compiler build number
__COUNTER__	The compiler for each encountered __COUNTER__ declaration substitutes the counter value from 0 to N-1 where N is a number of uses in the code. The __COUNTER__ order is guaranteed when recompiling the source code with no changes. The __COUNTER__ value is calculated the following way: • the initial counter value is 0, • after each counter usage, its value is increased by 1, • first, the compiler expands all macros and templates into source code on-site, • a separate code is created for each template function specialization, • a separate code is created for each template class/structure specialization, • next, the compiler passes through the obtained source code in the defined order and replaces each detected __COUNTER__ usage with the current counter value. The example below shows how the compiler handles the source code and replaces all instances of __COUNTER__ it meets with sequentially increasing values.

Constant	Description
__RANDOM__	The compiler inserts a random ulong value for each __RANDOM__ declaration.

Example:

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- an example of information output at Expert Advisor initialization
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//--- set the interval between the timer events
   EventSetTimer(5);
//---
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- an example of information output at Expert Advisor deinitialization
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//---
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- information output at tick receipt
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
   test2();
//---
  }
//+------------------------------------------------------------------+
//| test1                                                            |
//+------------------------------------------------------------------+
void test1(string par)
  {
//--- information output inside the function
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__," par=",par);
  }
```

```
//+------------------------------------------------------------------+
//| test2                                                            |
//+------------------------------------------------------------------+
void test2()
  {
//--- information output inside the function
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
  }
//+------------------------------------------------------------------+
//| OnTimer event handler                                            |
//+------------------------------------------------------------------+
void OnTimer()
  {
//---
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
  }
```

The example for learning how to work with the __COUNTER__ macro

```
//--- create a macro for a quick display of the expression and its value in the journa
#define print(expr) Print(#expr,"=",expr)

//--- define the MACRO_COUNTER custom macro via the predefined __COUNTER__ macro
#define MACRO_COUNTER __COUNTER__

//--- set the input value of the variable using the __COUNTER__ macro
input int InpVariable = __COUNTER__;

//--- set the value of the global variable using the __COUNTER__ macro before definin
int ExtVariable = __COUNTER__;

//+------------------------------------------------------------------+
//| the function returns the __COUNTER__ value                       |
//+------------------------------------------------------------------+
int GlobalFunc(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| the template function returns the __COUNTER__ value              |
//+------------------------------------------------------------------+
template<typename T>
int GlobalTemplateFunc(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| the structure with the method returning __COUNTER__              |
//+------------------------------------------------------------------+
struct A
  {
   int               dummy;   // not used

   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
//+------------------------------------------------------------------+
//| the template structure with the method returning __COUNTER__     |
//+------------------------------------------------------------------+
template<typename T>
struct B
  {
   int               dummy;   // not used

   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
//+------------------------------------------------------------------+
//| the structure with the template method returning __COUNTER__     |
//+------------------------------------------------------------------+
struct C
  {
   int               dummy;   // not used

   template<typename T>
   int               Method(void)
     {
      return(__COUNTER__);
     }
  };
//+------------------------------------------------------------------+
//| the function #2, which returns the __COUNTER__ value             |
//+------------------------------------------------------------------+
int GlobalFunc2(void)
  {
   return(__COUNTER__);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
```

```
// __COUNTER__ in the macro and the variables
   print(MACRO_COUNTER);
   print(InpVariable);
   print(ExtVariable);

//--- __COUNTER__ in the functions
   print(GlobalFunc());
   print(GlobalFunc());                // the value is not changed
   print(GlobalTemplateFunc<int>());
   print(GlobalTemplateFunc<int>());   // the value is not changed
   print(GlobalTemplateFunc<double>());// the value has changed
   print(GlobalFunc2());
   print(GlobalFunc2());               // the value is not changed

// __COUNTER__ in the structure
   A a1, a2;
   print(a1.Method());
   print(a2.Method());                 // the value is not changed

// __COUNTER__ in the template structure
   B<int> b1, b2;
   B<double> b3;
   print(b1.Method());
   print(b2.Method());                 // the value is not changed
   print(b3.Method());                 // the value has changed

// __COUNTER__ in the structure with the template function
   C c1, c2;
   print(c1.Method<int>());
   print(c1.Method<double>());         // the value has changed
   print(c2.Method<int>());            // the same value as during the first c1.Metho

//--- let's have another look at __COUNTER__ in the macro and the global variable
   print(MACRO_COUNTER);  // the value has changed
   print(ExtGlobal2);
  }
//--- set the value of the global variable using the __COUNTER__ macro after defining
int ExtGlobal2 = __COUNTER__;
//+------------------------------------------------------------------+

/* Result
   __COUNTER__=3
   InpVariable=0
   ExtVariable=1
   GlobalFunc()=5
   GlobalFunc()=5
   GlobalTemplateFunc<int>()=8
   GlobalTemplateFunc<int>()=8
   GlobalTemplateFunc<double>()=9
```

```
    GlobalFunc2()=7
    GlobalFunc2()=7
    a1.Method()=6
    a2.Method()=6
    b1.Method()=10
    b2.Method()=10
    b3.Method()=11
    c1.Method<int>()=12
    c1.Method<double>()=13
    c2.Method<int>()=12
    __COUNTER__=4
    ExtGlobal2=2

*/
```

Mathematical Constants

Special constants containing values are reserved for some mathematical expressions. These constants can be used in any place of the program instead of calculating their values using mathematical functions.

Constant	Description	Value
M_E	e	2.71828182845904523536
M_LOG2E	log2(e)	1.44269504088896340736
M_LOG10E	log10(e)	0.434294481903251827651
M_LN2	ln(2)	0.693147180559945309417
M_LN10	ln(10)	2.30258509299404568402
M_PI	pi	3.14159265358979323846
M_PI_2	pi/2	1.57079632679489661923
M_PI_4	pi/4	0.785398163397448309616
M_1_PI	1/pi	0.318309886183790671538
M_2_PI	2/pi	0.636619772367581343076
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390
M_SQRT2	sqrt(2)	1.41421356237309504880
M_SQRT1_2	1/sqrt(2)	0.707106781186547524401

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the values of constants
   Print("M_E = ",DoubleToString(M_E,16));
   Print("M_LOG2E = ",DoubleToString(M_LOG2E,16));
   Print("M_LOG10E = ",DoubleToString(M_LOG10E,16));
   Print("M_LN2 = ",DoubleToString(M_LN2,16));
   Print("M_LN10 = ",DoubleToString(M_LN10,16));
   Print("M_PI = ",DoubleToString(M_PI,16));
   Print("M_PI_2 = ",DoubleToString(M_PI_2,16));
   Print("M_PI_4 = ",DoubleToString(M_PI_4,16));
   Print("M_1_PI = ",DoubleToString(M_1_PI,16));
   Print("M_2_PI = ",DoubleToString(M_2_PI,16));
   Print("M_2_SQRTPI = ",DoubleToString(M_2_SQRTPI,16));
   Print("M_SQRT2 = ",DoubleToString(M_SQRT2,16));
   Print("M_SQRT1_2 = ",DoubleToString(M_SQRT1_2,16));
  }
```

Numerical Type Constants

Each simple numerical type is intended for a certain type of tasks and allows optimizing the operation of a mql5-program when used correctly. For a better code readability and correct handling of calculation results, there are constants which allow to receive information about restrictions set to a certain type of simple data.

Constant	Description	Value
CHAR_MIN	Minimal value, which can be represented by char type	-128
CHAR_MAX	Maximal value, which can be represented by char type	127
UCHAR_MAX	Maximal value, which can be represented by uchar type	255
SHORT_MIN	Minimal value, which can be represented by short type	-32768
SHORT_MAX	Maximal value, which can be represented by short type	32767
USHORT_MAX	Maximal value, which can be represented by ushort type	65535
INT_MIN	Minimal value, which can be represented by int type	-2147483648
INT_MAX	Maximal value, which can be represented by int type	2147483647
UINT_MAX	Maximal value, which can be represented by uint type	4294967295
LONG_MIN	Minimal value, which can be represented by long type	-9223372036854775808
LONG_MAX	Maximal value, which can be represented by long type	9223372036854775807
ULONG_MAX	Maximal value, which can be represented by ulong type	18446744073709551615
DBL_MIN	Minimal positive value, which can be represented by double type	2.2250738585072014e-308
DBL_MAX	Maximal value, which can be represented by double type	1.7976931348623158e+308
DBL_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for double type)	2.2204460492503131e-016
DBL_DIG	Number of significant decimal digits for double type	15

Constant	Description	Value
DBL_MANT_DIG	Number of bits in a mantissa for double type	53
DBL_MAX_10_EXP	Maximal decimal value of exponent degree for double type	308
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	1024
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	(-307)
DBL_MIN_EXP	Minimal binary value of exponent degree for double type	(-1021)
FLT_MIN	Minimal positive value, which can be represented by float type	1.175494351e-38
FLT_MAX	Maximal value, which can be represented by float type	3.402823466e+38
FLT_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for float type)	1.192092896e-07
FLT_DIG	Number of significant decimal digits for float type	6
FLT_MANT_DIG	Number of bits in a mantissa for float type	24
FLT_MAX_10_EXP	Maximal decimal value of exponent degree for float type	38
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	128
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	-37
FLT_MIN_EXP	Minimal binary value of exponent degree for float type	(-125)

Example:

```
void OnStart()
  {
//--- print the constant values
   printf("CHAR_MIN = %d",CHAR_MIN);
   printf("CHAR_MAX = %d",CHAR_MAX);
   printf("UCHAR_MAX = %d",UCHAR_MAX);
   printf("SHORT_MIN = %d",SHORT_MIN);
   printf("SHORT_MAX = %d",SHORT_MAX);
   printf("USHORT_MAX = %d",USHORT_MAX);
   printf("INT_MIN = %d",INT_MIN);
   printf("INT_MAX = %d",INT_MAX);
   printf("UINT_MAX = %u",UINT_MAX);
   printf("LONG_MIN = %I64d",LONG_MIN);
   printf("LONG_MAX = %I64d",LONG_MAX);
   printf("ULONG_MAX = %I64u",ULONG_MAX);
   printf("EMPTY_VALUE = %.16e",EMPTY_VALUE);
   printf("DBL_MIN = %.16e",DBL_MIN);
   printf("DBL_MAX = %.16e",DBL_MAX);
   printf("DBL_EPSILON = %.16e",DBL_EPSILON);
   printf("DBL_DIG = %d",DBL_DIG);
   printf("DBL_MANT_DIG = %d",DBL_MANT_DIG);
   printf("DBL_MAX_10_EXP =  %d",DBL_MAX_10_EXP);
   printf("DBL_MAX_EXP = %d",DBL_MAX_EXP);
   printf("DBL_MIN_10_EXP = %d",DBL_MIN_10_EXP);
   printf("DBL_MIN_EXP = %d",DBL_MIN_EXP);
   printf("FLT_MIN = %.8e",FLT_MIN);
   printf("FLT_MAX = %.8e",FLT_MAX);
   printf("FLT_EPSILON = %.8e",FLT_EPSILON);
/*
   CHAR_MIN = -128
   CHAR_MAX = 127
   UCHAR_MAX = 255
   SHORT_MIN = -32768
   SHORT_MAX = 32767
   USHORT_MAX = 65535
   INT_MIN = -2147483648
   INT_MAX = 2147483647
   UINT_MAX = 4294967295
   LONG_MIN = -9223372036854775808
   LONG_MAX = 9223372036854775807
   ULONG_MAX = 18446744073709551615
   EMPTY_VALUE = 1.7976931348623157e+308
   DBL_MIN = 2.2250738585072014e-308
   DBL_MAX = 1.7976931348623157e+308
   DBL_EPSILON = 2.2204460492503131e-16
   DBL_DIG = 15
   DBL_MANT_DIG = 53
   DBL_MAX_10_EXP =  308
   DBL_MAX_EXP = 1024
   DBL_MIN_10_EXP = -307
   DBL_MIN_EXP = -1021
   FLT_MIN = 1.17549435e-38
   FLT_MAX = 3.40282347e+38
   FLT_EPSILON = 1.19209290e-07
*/
  }
```

Uninitialization Reason Codes

Uninitialization reason codes are returned by the UninitializeReason() function. The possible values are the following:

Constant	Value	Description
REASON_PROGRAM	0	Expert Advisor terminated its operation by calling the ExpertRemove() function
REASON_REMOVE	1	Program has been deleted from the chart
REASON_RECOMPILE	2	Program has been recompiled
REASON_CHARTCHANGE	3	Symbol or chart period has been changed
REASON_CHARTCLOSE	4	Chart has been closed
REASON_PARAMETERS	5	Input parameters have been changed by a user
REASON_ACCOUNT	6	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings
REASON_TEMPLATE	7	A new template has been applied
REASON_INITFAILED	8	This value means that OnInit() handler has returned a nonzero value
REASON_CLOSE	9	Terminal has been closed

The uninitialization reason code is also passed as a parameter of the predetermined function OnDeinit(const int reason).

Example:

```
//+------------------------------------------------------------------+
//| get text description                                             |
//+------------------------------------------------------------------+
string getUninitReasonText(int reasonCode)
  {
   string text="";
//---
   switch(reasonCode)
     {
      case REASON_ACCOUNT:
         text="Account was changed";break;
      case REASON_CHARTCHANGE:
         text="Symbol or timeframe was changed";break;
      case REASON_CHARTCLOSE:
         text="Chart was closed";break;
      case REASON_PARAMETERS:
         text="Input-parameter was changed";break;
      case REASON_RECOMPILE:
         text="Program "+__FILE__+" was recompiled";break;
      case REASON_REMOVE:
         text="Program "+__FILE__+" was removed from chart";break;
      case REASON_TEMPLATE:
         text="New template was applied to chart";break;
      default:text="Another reason";
     }
//---
   return text;
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- The first way to get the uninitialization reason code
   Print(__FUNCTION__,"_Uninitalization reason code = ",reason);
    //--- The second way to get the uninitialization reason code
   Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason));
  }
```

Checking Object Pointer

The CheckPointer() function is used for checking the type of the object pointer. The function returns a value of the ENUM_POINTER_TYPE enumeration. If an incorrect pointer is used, the program execution will be immediately terminated.

Objects created by the new() operator are of POINTER_DYNAMIC type. The delete() operator can and should be used only for such pointers.

All other pointers are of POINTER_AUTOMATIC type, which means that this object has been created automatically by the mql5 program environment. Such objects are deleted automatically after being used.

ENUM_POINTER_TYPE

Constant	Description
POINTER_INVALID	Incorrect pointer
POINTER_DYNAMIC	Pointer of the object created by the new() operator
POINTER_AUTOMATIC	Pointer of any objects created automatically (not using new())

See also

Runtime errors, Object Delete Operator delete, CheckPointer

Other Constants

The CLR_NONE constant is used to outline the absence of color, it means that the graphical object or graphical series of an indicator will not be plotted. This constant was not included into the Web-color constants list, but it can be applied everywhere where the color arguments are required.

The INVALID_HANDLE constant can be used for checking file handles (see FileOpen() and FileFindFirst()).

Constant	Description	Value
CHARTS_MAX	The maximum possible number of simultaneously open charts in the terminal	100
clrNONE	Absence of color	-1
EMPTY_VALUE	Empty value in an indicator buffer	DBL_MAX
INVALID_HANDLE	Incorrect handle	-1
IS_DEBUG_MODE	Flag that a mq5-program operates in debug mode	non zero in debug mode, otherwise zero
IS_PROFILE_MODE	Flag that a mq5-program operates in profiling mode	non zero in profiling mode, otherwise zero
NULL	Zero for any types	0
WHOLE_ARRAY	Means the number of items remaining until the end of the array, i.e., the entire array will be processed	-1
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	-1

The EMPTY_VALUE constant usually corresponds to the values of indicators that are not shown in the chart. For example, for built-in indicator Standard Deviation with a period of 20, the line for the first 19 bars in the history is not shown in the chart. If you create a handle of this indicator with the iStdDev() function and copy it to an array of indicator values for these bars through CopyBuffer(), then these values will be equal to EMPTY_VALUE.

You can choose to specify for a custom indicator your own empty value of the indicator, when the indicator shouldn't be drawn in the chart. Use the PlotIndexSetDouble() function with the PLOT_EMPTY_VALUE modifier.

The NULL constant can be assigned to a variable of any simple type or to an object structure or class pointer. The NULL assignment for a string variable means the full deinitialization of this variable.

The WRONG_VALUE constant is intended for cases, when it is necessary to return value of an enumeration, and this must be a wrong value. For example, when we need to inform that a return value is a value from this enumeration. Let's consider as an example some function CheckLineStyle(), which returns the line style for an object, specified by its name. If at style check by

ObjectGetInteger() the result is true, a value from ENUM_LINE_STYLE is returned; otherwise WRONG_VALUE is returned.

```
void OnStart()
  {
   if(CheckLineStyle("MyChartObject")==WRONG_VALUE)
      printf("Error line style getting.");
  }
//+------------------------------------------------------------------+
//| returns the line style for an object specified by its name       |
//+------------------------------------------------------------------+
ENUM_LINE_STYLE CheckLineStyle(string name)
  {
   long style;
//---
   if(ObjectGetInteger(0,name,OBJPROP_STYLE,0,style))
      return((ENUM_LINE_STYLE)style);
   else
      return(WRONG_VALUE);
  }
```

The WHOLE_ARRAY constant is intended for functions that require specifying the number of elements in processed arrays:

- ArrayCopy();
- ArrayMinimum();
- ArrayMaximum();
- FileReadArray();
- FileWriteArray().

If you want to specify that all the array values from a specified position till the end must be processed, you should specify just the WHOLE_ARRAY value.

IS_PROFILE_MODE constant allows changing a program operation for correct data collection in the profiling mode. Profiling allows measuring the execution time of the individual program fragments (usually comprising functions), as well as calculating the number of such calls. Sleep() function calls can be disabled to determine the execution time in the profiling mode, like in this example:

```
//--- Sleep can greatly affect (change) profiling result
if(!IS_PROFILE_MODE) Sleep(100); // disabling Sleep() call in the profiling mode
```

IS_PROFILE_MODE constant value is set by the compiler during the compilation, while it is set to zero in conventional mode. When launching a program in the profiling mode, a special compilation is performed and IS_PROFILE_MODE is replaced with a non-zero value.

The IS_DEBUG_MODE constant can be useful when you need to slightly change the operation of a mql5 program in the debugging mode. For example, in debug mode you may need to display additional debugging information in the terminal log or create additional graphical objects in a chart.

The following example creates a Label object and sets its description and color depending on the script running mode. In order to run a script in the debug mode from MetaEditor, press F5. If you run the script from the browser window in the terminal, then the color and text of the object Label will be different.

Example:

```
//+------------------------------------------------------------------+
//|                                             Check_DEBUG_MODE.mq5 |
//|                       Copyright © 2009, MetaQuotes Software Corp. |
//|                                            https://www.metaquotes.net |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="invisible_label";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object",label_name,"not found. Error code = ",GetLastError());
      //--- create Label
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- set X coordinate
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- set Y coordinate
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      ResetLastError();
      if(IS_DEBUG_MODE) // debug mode
        {
         //--- show message about the script execution mode
         ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
         //--- set text color to red
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
            Print("Unable to set the color. Error",GetLastError());
        }
      else              // operation mode
        {
         ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
         //--- set text color to invisible
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
            Print("Unable to set the color. Error ",GetLastError());
        }
      ChartRedraw();
      DebugBreak();    // here termination will occur, if we are in debug mode
     }
  }
```

Crypt Methods

The ENUM_CRYPT_METHOD enumeration is used to specify the data transformation method, used in CryptEncode() and CryptDecode() functions.

ENUM_CRYPT_METHOD

Constant	Description
CRYPT_BASE64	BASE64
CRYPT_AES128	AES encryption with 128 bit key (16 bytes)
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)
CRYPT_DES	DES encryption with 56 bit key (7 bytes)
CRYPT_HASH_SHA1	SHA1 HASH calculation
CRYPT_HASH_SHA256	SHA256 HASH calculation
CRYPT_HASH_MD5	MD5 HASH calculation
CRYPT_ARCH_ZIP	ZIP archives

See also

DebugBreak, Executed MQL5 program properties, CryptEncode(), CryptDecode()

Data Structures

MQL5 Language offers 12 predefined structures:

- MqlDateTime is intended for working with date and time;
- MqlParam can send input parameters when creating a handle of the indicator using the IndicatorCreate() function;
- MqlRates is intended for manipulating the historical data, it contains information about the price, volume and spread;
- MqlBookInfo is intended for obtaining information about the Depth of Market;
- MqlTradeRequest is used for creating a trade request for trade operations;
- MqlTradeCheckResult is intended for checking the prepared trade request before sending it;
- MqlTradeResult contains a trade server reply to a trade request, sent by OrderSend() function;
- MqlTradeTransaction contains description of a trade transaction;
- MqlTick is designed for fast retrieval of the most requested information about current prices.
- Economic calendar structures are used to obtain data on the economic calendar events sent to the MetaTrader 5 platform in real time. Economic calendar functions allow analyzing macroeconomic parameters immediately after new reports are released, since relevant values are broadcast directly from the source with no delay.

MqlDateTime

The date type structure contains eight fields of the int type:

```
struct MqlDateTime
  {
   int year;           // Year
   int mon;            // Month
   int day;            // Day
   int hour;           // Hour
   int min;            // Minutes
   int sec;            // Seconds
   int day_of_week;    // Day of week (0-Sunday, 1-Monday, ... ,6-Saturday)
   int day_of_year;    // Day number of the year (January 1st is assigned the number
  };
```

Note

The day number of the year day_of_year for the leap year, since March, will differ from a number of the corresponding day for a non-leap year.

Example:

```
void OnStart()
  {
//---
   datetime date1=D'2008.03.01';
   datetime date2=D'2009.03.01';

   MqlDateTime str1,str2;
   TimeToStruct(date1,str1);
   TimeToStruct(date2,str2);
   printf("%02d.%02d.%4d, day of year = %d",str1.day,str1.mon,
          str1.year,str1.day_of_year);
   printf("%02d.%02d.%4d, day of year = %d",str2.day,str2.mon,
          str2.year,str2.day_of_year);
  }
/*  Result:
   01.03.2008, day of year = 60
   01.03.2009, day of year = 59
*/
```

See also

TimeToStruct, Structures and Classes

The Structure of Input Parameters of Indicators (MqlParam)

The MqlParam structure has been specially designed to provide input parameters when creating the handle of a technical indicator using the IndicatorCreate() function.

```
struct MqlParam
  {
   ENUM_DATATYPE     type;                    // type of the input parameter, value of
   long              integer_value;           // field to store an integer type
   double            double_value;            // field to store a double type
   string            string_value;            // field to store a string type
  };
```

All input parameters of an indicator are transmitted in the form of an array of the MqlParam type, the *type* field of each element of this array specifies the type of data transmitted by the element. The indicator values must be first placed in the appropriate fields for each element (in *integer_value*, in *double_value* or *string_value*) depending on what value of ENUM_DATATYPE enumeration is specified in the *type* field.

If the IND_CUSTOM value is passed third as the indicator type to the IndicatorCreate() function, the first element of the array of input parameters must have the *type* field with the value of TYPE_STRING from the ENUM_DATATYPE enumeration, and the *string_value* field must contain the name of the custom indicator.

MqlRates

This structure stores information about the prices, volumes and spread.

```
struct MqlRates
  {
   datetime time;         // Period start time
   double   open;         // Open price
   double   high;         // The highest price of the period
   double   low;          // The lowest price of the period
   double   close;        // Close price
   long     tick_volume;  // Tick volume
   int      spread;       // Spread
   long     real_volume;  // Trade volume
  };
```

Example:

```
void OnStart()
  {
   MqlRates rates[];
   int copied=CopyRates(NULL,0,0,100,rates);
   if(copied<=0)
      Print("Error copying price data ",GetLastError());
   else Print("Copied ",ArraySize(rates)," bars");
  }
```

See also

CopyRates, Access to timeseries

MqlBookInfo

It provides information about the market depth data.

```
struct MqlBookInfo
  {
   ENUM_BOOK_TYPE    type;             // Order type from ENUM_BOOK_TYPE enumeration
   double            price;            // Price
   long              volume;           // Volume
   double            volume_real;      // Volume with greater accuracy
  };
```

Note

The MqlBookInfo structure is predefined, thus it doesn't require any declaration and description. To use the structure, just declare a variable of this type.

The DOM is available only for some symbols.

Example:

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo about ",Symbol());
     }
   else
     {
      Print("Failed to receive DOM for the symbol ",Symbol());
     }
```

See also

MarketBookAdd, MarketBookRelease, MarketBookGet, Trade Orders in DOM, Data Types

The Trade Request Structure (MqlTradeRequest)

Interaction between the client terminal and a trade server for executing the order placing operation is performed by using trade requests. The trade request is represented by the special predefined structure of MqlTradeRequest type, which contain all the fields necessary to perform trade deals. The request processing result is represented by the structure of MqlTradeResult type.

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;           // Trade operation type
   ulong                         magic;            // Expert Advisor ID (magic number)
   ulong                         order;            // Order ticket
   string                        symbol;           // Trade symbol
   double                        volume;           // Requested volume for a deal in
   double                        price;            // Price
   double                        stoplimit;        // StopLimit level of the order
   double                        sl;               // Stop Loss level of the order
   double                        tp;               // Take Profit level of the order
   ulong                         deviation;        // Maximal possible deviation from
   ENUM_ORDER_TYPE               type;             // Order type
   ENUM_ORDER_TYPE_FILLING       type_filling;     // Order execution type
   ENUM_ORDER_TYPE_TIME          type_time;        // Order expiration type
   datetime                      expiration;       // Order expiration time (for the
   string                        comment;          // Order comment
   ulong                         position;         // Position ticket
   ulong                         position_by;      // The ticket of an opposite posit
  };
```

Fields description

Field	Description
action	Trade operation type. Can be one of the ENUM_TRADE_REQUEST_ACTIONS enumeration values.
magic	Expert Advisor ID. It allows organizing analytical processing of trade orders. Each Expert Advisor can set its own unique ID when sending a trade request.
order	Order ticket. It is used for modifying pending orders.
symbol	Symbol of the order. It is not necessary for order modification and position close operations.
volume	Requested order volume in lots. Note that the real volume of a deal will depend on the order execution type.
price	Price, reaching which the order must be executed. Market orders of symbols, whose execution type is "Market Execution" (SYMBOL_TRADE_EXECUTION_MARKET), of TRADE_ACTION_DEAL type, do not require specification of price.

Field	Description
stoplimit	The price value, at which the Limit pending order will be placed, when price reaches the *price* value (this condition is obligatory). Until then the pending order is not placed.
sl	Stop Loss price in case of the unfavorable price movement
tp	Take Profit price in the case of the favorable price movement
deviation	The maximal price deviation, specified in points
type	Order type. Can be one of the ENUM_ORDER_TYPE enumeration values.
type_filling	Order execution type. Can be one of the enumeration ENUM_ORDER_TYPE_FILLING values.
type_time	Order expiration type. Can be one of the enumeration ENUM_ORDER_TYPE_TIME values.
expiration	Order expiration time (for orders of ORDER_TIME_SPECIFIED type)
comment	Order comment
position	Ticket of a position. Should be filled in when a position is modified or closed to identify the position. As a rule it is equal to the ticket of the order, based on which the position was opened.
position_by	Ticket of an opposite position. Used when a position is closed by an opposite one open for the same symbol in the opposite direction.

When modifying or closing a position in the hedging system, make sure to specify its ticket (MqlTradeRequest::position). The ticket can also be specified in the netting system, though a position is identified by the symbol name.

For sending orders to perform trade operations it is necessary to use the OrderSend() function. For each trade operation it is necessary to specify obligatory fields; optional fields also may be filled. There are seven possible cases to send a trade order:

Request Execution

This is a trade order to open a position in the Request Execution mode (trade upon requested prices). It requires to specify the following 9 fields:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Instant Execution

This is a trade order to open a position in the Instant Execution mode (trade by current prices). It requires specification of the following 9 fields:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Market Execution

This is a trade order to open a position in the Market Execution mode. It requires to specify the following 5 fields:

- action
- symbol
- volume
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Exchange Execution

This is a trade order to open a position in the Exchange Execution mode. It requires to specify the following 5 fields:

- action
- symbol
- volume
- type
- type_filling

Also it is possible to specify the "magic" and "comment" field values.

Example of the TRADE_ACTION_DEAL trade operation for opening a Buy position:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Opening Buy position                                             |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters of request
   request.action   =TRADE_ACTION_DEAL;                     // type of trade operatior
   request.symbol   =Symbol();                              // symbol
   request.volume   =0.1;                                   // volume of 0.1 lot
   request.type     =ORDER_TYPE_BUY;                        // order type
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // price for opening
   request.deviation=5;                                     // allowed deviation from
   request.magic    =EXPERT_MAGIC;                          // MagicNumber of the orde
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // if unable to send the r
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_DEAL trade operation for opening a Sell position:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Opening Sell position                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters of request
   request.action   =TRADE_ACTION_DEAL;                     // type of trade operatior
   request.symbol   =Symbol();                              // symbol
   request.volume   =0.2;                                   // volume of 0.2 lot
   request.type     =ORDER_TYPE_SELL;                       // order type
   request.price    =SymbolInfoDouble(Symbol(),SYMBOL_BID); // price for opening
   request.deviation=5;                                     // allowed deviation from
   request.magic    =EXPERT_MAGIC;                          // MagicNumber of the orde
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());     // if unable to send the r
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Example of the TRADE_ACTION_DEAL trade operation for closing positions:

```
#define EXPERT_MAGIC 123456   // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Closing all positions                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
   for(int i=total-1; i>=0; i--)
     {
      //--- parameters of the order
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      double volume=PositionGetDouble(POSITION_VOLUME);
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- output information about the position
      PrintFormat("#%I64u %s  %s  %.2f  %s [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  magic);
      //--- if the MagicNumber matches
      if(magic==EXPERT_MAGIC)
        {
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action   =TRADE_ACTION_DEAL;        // type of trade operation
         request.position =position_ticket;          // ticket of the position
         request.symbol   =position_symbol;          // symbol
         request.volume   =volume;                   // volume of the position
         request.deviation=5;                        // allowed deviation from the pri
         request.magic    =EXPERT_MAGIC;             // MagicNumber of the position
         //--- set the price and order type depending on the position type
         if(type==POSITION_TYPE_BUY)
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
            request.type =ORDER_TYPE_SELL;
           }
         else
           {
            request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
            request.type =ORDER_TYPE_BUY;
           }
         //--- output information about the closure
         PrintFormat("Close #%I64d %s %s",position_ticket,position_symbol,EnumToString
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //---
        }
     }
  }
//+------------------------------------------------------------------+
```

SL & TP Modification

Trade order to modify the StopLoss and/or TakeProfit price levels. It requires to specify the following 4 fields:

- action
- symbol
- sl
- tp
- position

Example of the TRADE_ACTION_SLTP trade operation for modifying the Stop Loss and Take Profit values of an open position:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Modification of Stop Loss and Take Profit of position            |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request;
   MqlTradeResult  result;
   int total=PositionsTotal(); // number of open positions
//--- iterate over all open positions
   for(int i=0; i<total; i++)
     {
      //--- parameters of the order
      ulong  position_ticket=PositionGetTicket(i);// ticket of the position
      string position_symbol=PositionGetString(POSITION_SYMBOL); // symbol
      int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // number
      ulong  magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber of the position
      double volume=PositionGetDouble(POSITION_VOLUME);    // volume of the position
      double sl=PositionGetDouble(POSITION_SL);  // Stop Loss of the position
      double tp=PositionGetDouble(POSITION_TP);  // Take Profit of the position
      ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
      //--- output information about the position
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  position_ticket,
                  position_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         //--- calculate the current price levels
         double price=PositionGetDouble(POSITION_PRICE_OPEN);
         double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
         double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
         int    stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_
         double price_level;
         //--- if the minimum allowed offset distance in points from the current clos
         if(stop_level<=0)
            stop_level=150; // set the offset distance of 150 points from the current
         else
            stop_level+=50; // set the offset distance to (SYMBOL_TRADE_STOPS_LEVEL +

         //--- calculation and rounding of the Stop Loss and Take Profit values
         price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
         if(type==POSITION_TYPE_BUY)
           {
            sl=NormalizeDouble(bid-price_level,digits);
            tp=NormalizeDouble(bid+price_level,digits);
           }
         else
           {
            sl=NormalizeDouble(ask+price_level,digits);
            tp=NormalizeDouble(ask-price_level,digits);
           }
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action  =TRADE_ACTION_SLTP; // type of trade operation
         request.position=position_ticket;   // ticket of the position
         request.symbol=position_symbol;     // symbol
         request.sl      =sl;                // Stop Loss of the position
         request.tp      =tp;                // Take Profit of the position
         request.magic=EXPERT_MAGIC;         // MagicNumber of the position
         //--- output information about the modification
         PrintFormat("Modify #%I64d %s %s",position_ticket,position_symbol,EnumToStri
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send t
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

Pending Order

Trade order to place a pending order. It requires to specify the following 11 fields:

- action
- symbol
- volume
- price
- stoplimit
- sl
- tp
- type

- type_filling
- type_time
- expiration

Also it is possible to specify the "magic" and "comment" field values.

Example of the TRADE_ACTION_PENDING trade operation for placing a pending order:

```
#property description "Example of placing pending orders"
#property script_show_inputs
#define EXPERT_MAGIC 123456                             // MagicNumber of the expert
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;   // order type
//+------------------------------------------------------------------+
//| Placing pending orders                                           |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
//--- parameters to place a pending order
   request.action   =TRADE_ACTION_PENDING;                             // type of trac
   request.symbol   =Symbol();                                         // symbol
   request.volume   =0.1;                                              // volume of 0.
   request.deviation=2;                                                // allowed devi
   request.magic    =EXPERT_MAGIC;                                     // MagicNumber
   int offset = 50;                                                    // offset from
   double price;                                                       // order trigge
   double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT);                // value of poi
   int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);                // number of de
   //--- checking the type of operation
   if(orderType==ORDER_TYPE_BUY_LIMIT)
     {
      request.type     =ORDER_TYPE_BUY_LIMIT;                          // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;        // price for op
      request.price    =NormalizeDouble(price,digits);                 // normalized o
     }
   else if(orderType==ORDER_TYPE_SELL_LIMIT)
     {
      request.type     =ORDER_TYPE_SELL_LIMIT;                          // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;         // price for o
      request.price    =NormalizeDouble(price,digits);                  // normalized
     }
   else if(orderType==ORDER_TYPE_BUY_STOP)
     {
      request.type =ORDER_TYPE_BUY_STOP;                                // order type
      price        =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // price for o
      request.price=NormalizeDouble(price,digits);                      // normalized
     }
   else if(orderType==ORDER_TYPE_SELL_STOP)
     {
      request.type     =ORDER_TYPE_SELL_STOP;                           // order type
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point;         // price for o
      request.price    =NormalizeDouble(price,digits);                  // normalized
     }
   else Alert("This example is only for placing pending orders");   // if not pending
//--- send the request
   if(!OrderSend(request,result))
      PrintFormat("OrderSend error %d",GetLastError());                 // if unable t
//--- information about the operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
  }
//+------------------------------------------------------------------+
```

Modify Pending Order

Trade order to modify the prices of a pending order. It requires to specify the following 7 fields:

- action

- order
- price
- sl
- tp
- type_time
- expiration

Example of the TRADE_ACTION_MODIFY trade operation for modifying the price levels of pending orders:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
//+------------------------------------------------------------------+
//| Modification of pending orders                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
   for(int i=0; i<total; i++)
     {
      //--- parameters of the order
      ulong  order_ticket=OrderGetTicket(i);                             // order tic
      string order_symbol=Symbol();                                      // symbol
      int    digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS); // number of
      ulong  magic=OrderGetInteger(ORDER_MAGIC);                         // MagicNumb
      double volume=OrderGetDouble(ORDER_VOLUME_CURRENT);                // current vo
      double sl=OrderGetDouble(ORDER_SL);                                // current St
      double tp=OrderGetDouble(ORDER_TP);                                // current Ta
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // type of th
      int offset = 50;                                                   // offset fro
      double price;                                                      // order trig
      double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT);          // value of
      //--- output information about the order
      PrintFormat("#%I64u %s  %s  %.2f  %s  sl: %s  tp: %s  [%I64d]",
                  order_ticket,
                  order_symbol,
                  EnumToString(type),
                  volume,
                  DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                  DoubleToString(sl,digits),
                  DoubleToString(tp,digits),
                  magic);
      //--- if the MagicNumber matches, Stop Loss and Take Profit are not defined
      if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
         request.action=TRADE_ACTION_MODIFY;                           // type of trad
         request.order = OrderGetTicket(i);                            // order ticket
         request.symbol   =Symbol();                                   // symbol
         request.deviation=5;                                          // allowed devi
        //--- setting the price level, Take Profit and Stop Loss of the order dependin
         if(type==ORDER_TYPE_BUY_LIMIT)
           {
            price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price    =NormalizeDouble(price,digits);               // normali
           }
         else if(type==ORDER_TYPE_SELL_LIMIT)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price    =NormalizeDouble(price,digits);                // normal
           }
         else if(type==ORDER_TYPE_BUY_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;
            request.tp = NormalizeDouble(price+offset*point,digits);
            request.sl = NormalizeDouble(price-offset*point,digits);
            request.price     =NormalizeDouble(price,digits);                 // normal
           }
         else if(type==ORDER_TYPE_SELL_STOP)
           {
           price = SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point;
            request.tp = NormalizeDouble(price-offset*point,digits);
            request.sl = NormalizeDouble(price+offset*point,digits);
            request.price     =NormalizeDouble(price,digits);                 // normal
           }
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send th
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
        }
     }
  }
//+------------------------------------------------------------------+
```

Delete Pending Order

Trade order to delete a pending order. It requires to specify the following 2 fields:

- action
- order

Example of the TRADE_ACTION_REMOVE trade operation for deleting pending orders:

```
#define EXPERT_MAGIC 123456  // MagicNumber of the expert
```

```
//+------------------------------------------------------------------+
//| Deleting pending orders                                          |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the trade request and result of trade request
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   int total=OrdersTotal(); // total number of placed pending orders
//--- iterate over all placed pending orders
   for(int i=total-1; i>=0; i--)
     {
      ulong  order_ticket=OrderGetTicket(i);                   // order ticket
      ulong  magic=OrderGetInteger(ORDER_MAGIC);               // MagicNumber of the
      //--- if the MagicNumber matches
      if(magic==EXPERT_MAGIC)
        {
         //--- zeroing the request and result values
         ZeroMemory(request);
         ZeroMemory(result);
         //--- setting the operation parameters
         request.action=TRADE_ACTION_REMOVE;                   // type of trade operat
         request.order = order_ticket;                         // order ticket
         //--- send the request
         if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError());  // if unable to send t
         //--- information about the operation
         PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,
        }
     }
  }
//+------------------------------------------------------------------+
```

See also

Structures and Classes, Trade Functions, Order Properties

The Structure of Results of a Trade Request Check (MqlTradeCheckResult)

Before sending a request for a trade operation to a trade server, it is recommended to check it. The check is performed using the OrderCheck() function, to which the checked request and a variable of the MqlTradeCheckResult structure type are passed. The check result will be written to this variable.

```
struct MqlTradeCheckResult
  {
   uint         retcode;             // Reply code
   double       balance;             // Balance after the execution of the deal
   double       equity;              // Equity after the execution of the deal
   double       profit;              // Floating profit
   double       margin;              // Margin requirements
   double       margin_free;         // Free margin
   double       margin_level;        // Margin level
   string       comment;             // Comment to the reply code (description of the
  };
```

Description of Fields

Field	Description
retcode	Return code
balance	Balance value that will be after the execution of the trade operation
equity	Equity value that will be after the execution of the trade operation
profit	Value of the floating profit that will be after the execution of the trade operation
margin	Margin required for the trade operation
margin_free	Free margin that will be left after the execution of the trade operation
margin_level	Margin level that will be set after the execution of the trade operation
comment	Comment to the reply code, error description

See also

Trade Request Structure, Structure for Current Prices, OrderSend, OrderCheck

The Structure of a Trade Request Result (MqlTradeResult)

As result of a trade request, a trade server returns data about the trade request processing result as a special predefined structure of MqlTradeResult type.

```
struct MqlTradeResult
  {
   uint     retcode;          // Operation return code
   ulong    deal;             // Deal ticket, if it is performed
   ulong    order;            // Order ticket, if it is placed
   double   volume;           // Deal volume, confirmed by broker
   double   price;            // Deal price, confirmed by broker
   double   bid;              // Current Bid price
   double   ask;              // Current Ask price
   string   comment;          // Broker comment to operation (by default it is filled
   uint     request_id;       // Request ID set by the terminal during the dispatch
   int      retcode_external; // Return code of an external trading system
  };
```

Fields description

Field	Description
retcode	Return code of a trade server
deal	Deal ticket, if a deal has been performed. It is available for a trade operation of TRADE_ACTION_DEAL type
order	Order ticket, if a ticket has been placed. It is available for a trade operation of TRADE_ACTION_PENDING type
volume	Deal volume, confirmed by broker. It depends on the order filling type
price	Deal price, confirmed by broker. It depends on the *deviation* field of the trade request and/or on the trade operation
bid	The current market Bid price (requote price)
ask	The current market Ask price (requote price)
comment	The broker comment to operation (by default it is filled by description of trade server return code)
request_id	Request ID set by the terminal when sending to the trade server
retcode_external	The code of the error returned by an external trading system. The use and types of these errors depend on the broker and the external trading system, to which trading operations are sent.

The trade operation result is returned to a variable of the MqlTradeResult type, which is passed as the second parameter to OrderSend() to perform trade operations.

The terminal fixes request ID in request_id field when sending it to the trade server using OrdersSend() and OrderSendAsync() functions. The terminal receives messages about performed transactions from the trade server and submits them for processing by OnTradeTransaction() function containing the following components as parameters:

- description of the trade transaction in MqlTradeTransaction structure;
- description of the trade request sent from OrderSend() or OrdersSendAsync() function. Request ID is sent by the terminal to the trade server, while the request itself and its request_id are stored in the terminal memory;
- the trade request execution result as MqlTradeResult structure with request_id field containing ID of this request.

OnTradeTransaction() function receives three input parameters but the last two should be analyzed only for transactions having TRADE_TRANSACTION_REQUEST type. In all other cases, data on the trade request and its execution result are not filled. Example of parameters analysis can be found at Structure of a Trade Request.

Setting request_id by the terminal for the trade request when sending it to the server is mainly introduced for working with OrderSendAsync() asynchronous function. This identifier allows to associate the performed action (OrderSend or OrderSendAsync functions call) with the result of this action sent to OnTradeTransaction().

Example:

```
//+------------------------------------------------------------------+
//| Sending a trade request with the result processing               |
//+------------------------------------------------------------------+
bool MyOrderSend(MqlTradeRequest request,MqlTradeResult result)
  {
//--- reset the last error code to zero
   ResetLastError();
//--- send request
   bool success=OrderSend(request,result);
//--- if the result fails - try to find out why
   if(!success)
     {
      int answer=result.retcode;
      Print("TradeLog: Trade request failed. Error = ",GetLastError());
      switch(answer)
        {
         //--- requote
         case 10004:
           {
            Print("TRADE_RETCODE_REQUOTE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- order is not accepted by the server
         case 10006:
           {
            Print("TRADE_RETCODE_REJECT");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- invalid price
         case 10015:
           {
            Print("TRADE_RETCODE_INVALID_PRICE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- invalid SL and/or TP
         case 10016:
           {
            Print("TRADE_RETCODE_INVALID_STOPS");
            Print("request.sl = ",request.sl," request.tp = ",request.tp);
            Print("result.ask = ",result.ask," result.bid = ",result.bid);
            break;
           }
         //--- invalid volume
         case 10014:
           {
            Print("TRADE_RETCODE_INVALID_VOLUME");
            Print("request.volume = ",request.volume,"   result.volume = ",
                  result.volume);
            break;
           }
         //--- not enough money for a trade operation
         case 10019:
           {
            Print("TRADE_RETCODE_NO_MONEY");
            Print("request.volume = ",request.volume,"   result.volume = ",
                  result.volume,"   result.comment = ",result.comment);
            break;
           }
         //--- some other reason, output the server response code
         default:
           {
            Print("Other answer = ",answer);
           }
        }
      //--- notify about the unsuccessful result of the trade request by returning fal
      return(false);
     }
//--- OrderSend() returns true - repeat the answer
   return(true);
  }
```

Structure of a Trade Transaction (MqlTradeTransaction)

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using OrderSend and OrderSendAsync functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions.

Special OnTradeTransaction() handler is provided in MQL5 to get trade transactions applied to an account. The first parameter of the handler gets MqlTradeTransaction structure describing trade transactions.

```
struct MqlTradeTransaction
  {
   ulong                         deal;             // Deal ticket
   ulong                         order;            // Order ticket
   string                        symbol;           // Trade symbol name
   ENUM_TRADE_TRANSACTION_TYPE   type;             // Trade transaction type
   ENUM_ORDER_TYPE               order_type;       // Order type
   ENUM_ORDER_STATE              order_state;      // Order state
   ENUM_DEAL_TYPE                deal_type;        // Deal type
   ENUM_ORDER_TYPE_TIME          time_type;        // Order type by action period
   datetime                      time_expiration;  // Order expiration time
   double                        price;            // Price
   double                        price_trigger;    // Stop limit order activation pric
   double                        price_sl;         // Stop Loss level
   double                        price_tp;         // Take Profit level
   double                        volume;           // Volume in lots
   ulong                         position;         // Position ticket
   ulong                         position_by;      // Ticket of an opposite position
  };
```

Fields Description

Field	Description
deal	Deal ticket.
order	Order ticket.
symbol	The name of the trading symbol, for which transaction is performed.
type	Trade transaction type. The value can be one of ENUM_TRADE_TRANSACTION_TYPE enumeration values.
order_type	Trade order type. The value can be one of ENUM_ORDER_TYPE enumeration values.
order_state	Trade order state. The value can be one of ENUM_ORDER_STATE enumeration values.
deal_type	Deal type. The value can be one of ENUM_DEAL_TYPE enumeration values.
time_type	Order type upon expiration. The value can be one of ENUM_ORDER_TYPE_TIME values.
time_expiration	Pending order expiration term (for orders of ORDER_TIME_SPECIFIED and ORDER_TIME_SPECIFIED_DAY types).
price	Price. Depending on a trade transaction type, it may be a price of an order, a deal or a position.
price_trigger	Stop limit order stop (activation) price (ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT).
price_sl	Stop Loss price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
price_tp	Take Profit price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
volume	Volume in lots. Depending on a trade transaction type, it may indicate the current volume of an order, a deal or a position.
position	The ticket of the position affected by the transaction.
position_by	The ticket of the opposite position. Used when closing a position by an opposite one, i.e. by a position of the same symbol that was opened in the opposite direction.

The essential parameter for received transaction analysis is its type specified in **type** field. For example, if a transaction is of TRADE_TRANSACTION_REQUEST type (a result of handling a trade request by the server has been received), the structure has only only one field that is filled completely - **type**. Other fields are not analyzed. In this case, we may analyze two additional **request** and **result** parameters submitted to OnTradeTransaction() handler, as shown below.

Having data on a trading operation type, you can decide on the analysis of the current state of orders, positions and deals on a trading account. Remember that one trade request sent to the server from the terminal can generate several new transactions. The priority of their arrival at the terminal is not guaranteed.

MqlTradeTransaction structure is filled in different ways depending on a trade transaction type (ENUM_TRADE_TRANSACTION_TYPE):

TRADE_TRANSACTION_ORDER_* and TRADE_TRANSACTION_HISTORY_*

The following fields in MqlTradeTransaction structure are filled for trade transactions related to open orders handling (TRADE_TRANSACTION_ORDER_ADD, TRADE_TRANSACTION_ORDER_UPDATE and TRADE_TRANSACTION_ORDER_DELETE) and orders history (TRADE_TRANSACTION_HISTORY_ADD, TRADE_TRANSACTION_HISTORY_UPDATE, TRADE_TRANSACTION_HISTORY_DELETE):

- order - order ticket;
- symbol - order symbol name;
- type - trade transaction type;
- order_type - order type;
- orders_state - order current state;
- time_type - order expiration type;
- time_expiration - order expiration time (for orders having ORDER_TIME_SPECIFIED and ORDER_TIME_SPECIFIED_DAY expiration types);
- price - order price specified by a client;
- price_trigger - stop limit order stop price (only for ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT);
- price_sl - Stop Loss order price (filled, if specified in the order);
- price_tp - Take Profit order price (filled, if specified in the order);
- volume - order current volume (unfilled). Initial order volume can be found in the orders history using HistoryOrders* function.
- position - the ticket of the position that was opened, modified or closed as a result of order execution. It is only filled for market orders, not filled for TRADE_TRANSACTION_ORDER_ADD.
- position_by - the ticket of the opposite position. It is only filled for the close by orders (to close a position by an opposite one).

TRADE_TRANSACTION_DEAL_*

The following fields in MqlTradeTransaction structure are filled for trade transactions related to deals handling (TRADE_TRANSACTION_DEAL_ADD, TRADE_TRANSACTION_DEAL_UPDATE and TRADE_TRANSACTION_DEAL_DELETE):

- deal - deal ticket;
- order - order ticket, based on which a deal has been performed;
- symbol - deal symbol name;
- type - trade transaction type;
- deal_type - deal type;
- price - deal price;
- price_sl - Stop Loss price (filled, if specified in the order, based on which a deal has been performed);
- price_tp - Take Profit price (filled, if specified in the order, based on which a deal has been performed);
- volume - deal volume in lots.
- position - the ticket of the position that was opened, modified or closed as a result of deal execution.

- position_by - the ticket of the opposite position. It is only filled for the out by deals (closing a position by an opposite one).

TRADE_TRANSACTION_POSITION

The following fields in MqlTradeTransaction structure are filled for trade transactions related to changing the positions not connected with deals execution (TRADE_TRANSACTION_POSITION):

- symbol - position symbol name;
- type - trade transaction type;
- deal_type - position type (DEAL_TYPE_BUY or DEAL_TYPE_SELL);
- price - weighted average position open price;
- price_sl - Stop Loss price;
- price_tp - Take Profit price;
- volume - position volume in lots, if it has been changed.

> Position change (adding, changing or closing), as a result of a deal execution, does not lead to the occurrence of TRADE_TRANSACTION_POSITION transaction.

TRADE_TRANSACTION_REQUEST

Only one field in MqlTradeTransaction structure is filled for trade transactions describing the fact that a trade request has been processed by a server and processing result has been received (TRADE_TRANSACTION_REQUEST):

- type - trade transaction type;

> Only type field (trade transaction type) must be analyzed for such transactions. The second and third parameters of OnTradeTransaction function (request and result) must be analyzed for additional data.

Example:

```
input int MagicNumber=1234567;

//--- enable CTrade trading class and declare the variable of this class
#include <Trade\Trade.mqh>
CTrade trade;
//--- flags for installing and deleting the pending order
bool pending_done=false;
bool pending_deleted=false;
//--- pending order ticket will be stored here
ulong order_ticket;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set MagicNumber to mark all our orders
   trade.SetExpertMagicNumber(MagicNumber);
//--- trade requests will be sent in asynchronous mode using OrderSendAsync() functio
   trade.SetAsyncMode(true);
//--- initialize the variable by zero
```

```
   order_ticket=0;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---installing a pending order
   if(!pending_done)
     {
      double ask=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
      double buy_stop_price=NormalizeDouble(ask+1000*_Point,(int)SymbolInfoInteger(_Sy
      bool res=trade.BuyStop(0.1,buy_stop_price,_Symbol);
      //--- if BuyStop() function performed successfully
      if(res)
        {
         pending_done=true;
         //--- get a result of the request sending from ctrade
         MqlTradeResult trade_result;
         trade.Result(trade_result);
         //--- get request_id for the sent request
         uint request_id=trade_result.request_id;
         Print("Request has been sent to set a pending order. Request_ID=",request_id)
         //--- storing the order ticket (will be zero if using the asynchronous mode
         order_ticket=trade_result.order;
         //--- all is done, early exit from OnTick() handler
         return;
        }
     }
//--- delete the pending order
   if(!pending_deleted)
      //--- additional check
      if(pending_done && (order_ticket!=0))
        {
         //--- trying to delete the pending order
         bool res=trade.OrderDelete(order_ticket);
         Print("OrderDelete=",res);
         //--- when delete request is sent successfully
         if(res)
           {
            pending_deleted=true;
            //--- get the request execution result
            MqlTradeResult trade_result;
            trade.Result(trade_result);
            //--- take request ID from the result
            uint request_id=trade_result.request_id;
            //--- display in Journal
            Print("The request has been sent to delete a pending order #",order_ticket
                  ". Request_ID=",request_id,
                  "\r\n");
            //--- fix the order ticket from the request result
            order_ticket=trade_result.order;
           }
        }
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//--- get transaction type as enumeration value
   ENUM_TRADE_TRANSACTION_TYPE type=(ENUM_TRADE_TRANSACTION_TYPE)trans.type;
//--- if the transaction is the request handling result, only its name is displayed
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      Print(EnumToString(type));
      //--- display the handled request string name
      Print("------------RequestDescription\r\n",RequestDescription(request));
      //--- display request result description
      Print("------------ResultDescription\r\n",TradeResultDescription(result));
      //--- store the order ticket for its deletion at the next handling in OnTick()
      if(result.order!=0)
        {
         //--- delete this order by its ticket at the next OnTick() call
         order_ticket=result.order;
         Print(" Pending order ticket ",order_ticket,"\r\n");
        }
     }
   else // display the full description for transactions of another type
//--- display description of the received transaction in the Journal
      Print("------------TransactionDescription\r\n",TransactionDescription(trans));

//---
  }
//+------------------------------------------------------------------+
//| Returns transaction textual description                          |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans)
  {
//---
   string desc=EnumToString(trans.type)+"\r\n";
   desc+="Symbol: "+trans.symbol+"\r\n";
   desc+="Deal ticket: "+(string)trans.deal+"\r\n";
   desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
   desc+="Order ticket: "+(string)trans.order+"\r\n";
   desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
   desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
   desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
   desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
   desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
   desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
   desc+="Position: "+(string)trans.position+"\r\n";
   desc+="Position by: "+(string)trans.position_by+"\r\n";
//--- return the obtained string
   return desc;
  }
//+------------------------------------------------------------------+
//| Returns the trade request textual description                    |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request)
  {
//---
   string desc=EnumToString(request.action)+"\r\n";
   desc+="Symbol: "+request.symbol+"\r\n";
   desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
   desc+="Order ticket: "+(string)request.order+"\r\n";
   desc+="Order type: "+EnumToString(request.type)+"\r\n";
   desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
   desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
   desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
   desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
   desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
   desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
   desc+="Comment: "+request.comment+"\r\n";
//--- return the obtained string
   return desc;
  }
//+------------------------------------------------------------------+
//| Returns the textual description of the request handling result   |
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result)
  {
//---
   string desc="Retcode "+(string)result.retcode+"\r\n";
   desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
   desc+="Order ticket: "+(string)result.order+"\r\n";
```

```
    desc+="Deal ticket: "+(string)result.deal+"\r\n";
    desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
    desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
    desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
    desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
    desc+="Comment: "+result.comment+"\r\n";
//--- return the obtained string
    return desc;
  }
```

See also

Trade Transaction Types, OnTradeTransaction()

The Structure for Returning Current Prices (MqlTick)

This is a structure for storing the latest prices of the symbol. It is designed for fast retrieval of the most requested information about current prices.

```
struct MqlTick
  {
   datetime     time;          // Time of the last prices update
   double       bid;           // Current Bid price
   double       ask;           // Current Ask price
   double       last;          // Price of the last deal (Last)
   ulong        volume;        // Volume for the current Last price
   long         time_msc;      // Time of a price last update in milliseconds
   uint         flags;         // Tick flags
   double       volume_real;   // Volume for the current Last price with greater accur
  };
```

The variable of the MqlTick type allows obtaining values of Ask, Bid, Last and Volume within a single call of the SymbolInfoTick() function.

The parameters of each tick are filled in regardless of whether there are changes compared to the previous tick. Thus, it is possible to find out a correct price for any moment in the past without the need to search for previous values at the tick history. For example, even if only a Bid price changes during a tick arrival, the structure still contains other parameters as well, including the previous Ask price, volume, etc.

You can analyze the tick flags to find out what data have been changed exactly:

- TICK_FLAG_BID – tick has changed a Bid price
- TICK_FLAG_ASK – a tick has changed an Ask price
- TICK_FLAG_LAST – a tick has changed the last deal price
- TICK_FLAG_VOLUME – a tick has changed a volume
- TICK_FLAG_BUY – a tick is a result of a buy deal
- TICK_FLAG_SELL – a tick is a result of a sell deal

Example:

```
void OnTick()
  {
   MqlTick last_tick;
//---
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      Print(last_tick.time,": Bid = ",last_tick.bid,
            " Ask = ",last_tick.ask,"  Volume = ",last_tick.volume);
     }
   else Print("SymbolInfoTick() failed, error = ",GetLastError());
//---
  }
```

See also

Structures and Classes, CopyTicks(), SymbolInfoTick()

Economic Calendar structures

This section describes the structures for working with the economic calendar available directly in the MetaTrader platform. The economic calendar is a ready-made encyclopedia featuring descriptions of macroeconomic indicators, their release dates and degrees of importance. Relevant values of macroeconomic indicators are sent to the MetaTrader platform right at the moment of publication and are displayed on a chart as tags allowing you to visually track the required indicators by countries, currencies and importance.

Economic calendar functions allow conducting the auto analysis of incoming events according to custom importance criteria from a perspective of necessary countries/currency pairs.

Country descriptions are set by the MqlCalendarCountry structure. It is used in the CalendarCountryById() and CalendarCountries() functions

```
struct MqlCalendarCountry
  {
   ulong                               id;                    // country ID (ISO 3166-
   string                              name;                  // country text name (ir
   string                              code;                  // country code name (IS
   string                              currency;              // country currency code
   string                              currency_symbol;       // country currency symb
   string                              url_name;              // country name used in
  };
```

Event descriptions are set by the MqlCalendarEvent structure. It is used in the CalendarEventById(), CalendarEventByCountry() and CalendarEventByCurrency() functions

```
struct MqlCalendarEvent
  {
   ulong                               id;                    // event ID
   ENUM_CALENDAR_EVENT_TYPE            type;                  // event type from the
   ENUM_CALENDAR_EVENT_SECTOR          sector;                // sector an event is re
   ENUM_CALENDAR_EVENT_FREQUENCY       frequency;             // event frequency
   ENUM_CALENDAR_EVENT_TIMEMODE        time_mode;             // event time mode
   ulong                               country_id;            // country ID
   ENUM_CALENDAR_EVENT_UNIT            unit;                  // economic indicator va
   ENUM_CALENDAR_EVENT_IMPORTANCE      importance;            // event importance
   ENUM_CALENDAR_EVENT_MULTIPLIER      multiplier;            // economic indicator va
   uint                                digits;                // number of decimal pla
   string                              source_url;            // URL of a source where
   string                              event_code;            // event code
   string                              name;                  // event text name in th
  };
```

Event values are set by the MqlCalendarValue structure. It is used in the CalendarValueById(), CalendarValueHistoryByEvent(), CalendarValueHistory(), CalendarValueLastByEvent() and CalendarValueLast() functions

```
struct MqlCalendarValue
  {
   ulong                               id;                    // value ID
   ulong                               event_id;              // event ID
   datetime                            time;                  // event date and time
   datetime                            period;                // event reporting peri
   int                                 revision;              // revision of the publ
   long                                actual_value;          // actual value multipl
   long                                prev_value;            // previous value multip
   long                                revised_prev_value;    // revised previous valu
   long                                forecast_value;        // forecast value multip
   ENUM_CALENDAR_EVENT_IMPACT          impact_type;           // potential impact on t
  //--- functions checking the values
   bool                         HasActualValue(void) const;   // returns true if actua
   bool                         HasPreviousValue(void) const; // returns true if prev_
   bool                         HasRevisedValue(void) const;  // returns true if revis
   bool                         HasForecastValue(void) const; // returns true if forec
  //--- functions receiving the values
   double                       GetActualValue(void) const;   // returns actual_value
   double                       GetPreviousValue(void) const; // returns prev_value o
   double                       GetRevisedValue(void) const;  // returns revised_prev_
   double                       GetForecastValue(void) const; // returns forecast_valu
  };
```

The MqlCalendarValue structure provides methods for checking and setting values from the *actual_value*, *forecast_value*, *prev_value* and *revised_prev_value* fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in the field, then you should divide the value by 1,000,000 in order to get the desired value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

An example of handling calendar events:

```
//--- Create a structure to store calendar events with real values instead of integers
struct AdjustedCalendarValue
  {
   ulong                               id;                    // value ID
   ulong                               event_id;              // event ID
   datetime                            time;                  // event date and time
   datetime                            period;                // event reporting peri
   int                                 revision;              // revision of the publ
   double                              actual_value;          // actual value
   double                                 prev_value;            // previous value
   double                                 revised_prev_value;    // revised previous valu
   double                                 forecast_value;        // forecast value
   ENUM_CALENDAR_EVENT_IMPACT             impact_type;           // potential impact on t
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
//--- country code for EU (ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- get all EU event values
   MqlCalendarValue values[];
//--- set the boundaries of the interval we take the events from
   datetime date_from=D'01.01.2021';  // take all events from 2021
   datetime date_to=0;                // 0 means all known events, including the ones
//--- request EU event history since 2021
   if(!CalendarValueHistory(values, date_from, date_to, EU_code))
     {
      PrintFormat("Error! Failed to get events for country_code=%s", EU_code);
      PrintFormat("Error code: %d", GetLastError());
      return;
     }
   else
      PrintFormat("Received event values for country_code=%s: %d",
                  EU_code, ArraySize(values));
//--- reduce the size of the array for output to the Journal
   if(ArraySize(values)>5)
      ArrayResize(values, 5);
//--- output event values to the Journal as they are, without checking or converting t
   Print("Output calendar values as they are");
   ArrayPrint(values);

//--- check the field values and convert to actual values
//--- option 1 to check and get the values
   AdjustedCalendarValue values_adjusted_1[];
   int total=ArraySize(values);
   ArrayResize(values_adjusted_1, total);
//--- copy the values with checks and adjustments
   for(int i=0; i<total; i++)
     {
      values_adjusted_1[i].id=values[i].id;
      values_adjusted_1[i].event_id=values[i].event_id;
      values_adjusted_1[i].time=values[i].time;
      values_adjusted_1[i].period=values[i].period;
      values_adjusted_1[i].revision=values[i].revision;
      values_adjusted_1[i].impact_type=values[i].impact_type;
      //--- check values and divide by 1,000,000
      if(values[i].actual_value==LONG_MIN)
         values_adjusted_1[i].actual_value=double("nan");
      else
         values_adjusted_1[i].actual_value=values[i].actual_value/1000000.;

      if(values[i].prev_value==LONG_MIN)
         values_adjusted_1[i].prev_value=double("nan");
      else
         values_adjusted_1[i].prev_value=values[i].prev_value/1000000.;

      if(values[i].revised_prev_value==LONG_MIN)
         values_adjusted_1[i].revised_prev_value=double("nan");
      else
         values_adjusted_1[i].revised_prev_value=values[i].revised_prev_value/1000000

      if(values[i].forecast_value==LONG_MIN)
         values_adjusted_1[i].forecast_value=double("nan");
      else
         values_adjusted_1[i].forecast_value=values[i].forecast_value/1000000.;
     }
   Print("The first method to check and get calendar values");
   ArrayPrint(values_adjusted_1);

//--- option 2 to check and get the values
   AdjustedCalendarValue values_adjusted_2[];
   ArrayResize(values_adjusted_2, total);
//--- copy the values with checks and adjustments
   for(int i=0; i<total; i++)
     {
      values_adjusted_2[i].id=values[i].id;
      values_adjusted_2[i].event_id=values[i].event_id;
      values_adjusted_2[i].time=values[i].time;
      values_adjusted_2[i].period=values[i].period;
      values_adjusted_2[i].revision=values[i].revision;
      values_adjusted_2[i].impact_type=values[i].impact_type;
      //--- check and get values
      if(values[i].HasActualValue())
         values_adjusted_2[i].actual_value=values[i].GetActualValue();
      else
         values_adjusted_2[i].actual_value=double("nan");

      if(values[i].HasPreviousValue())
         values_adjusted_2[i].prev_value=values[i].GetPreviousValue();
      else
         values_adjusted_2[i].prev_value=double("nan");

      if(values[i].HasRevisedValue())
         values_adjusted_2[i].revised_prev_value=values[i].GetRevisedValue();
      else
         values_adjusted_2[i].revised_prev_value=double("nan");

      if(values[i].HasForecastValue())
         values_adjusted_2[i].forecast_value=values[i].GetForecastValue();
      else
         values_adjusted_2[i].forecast_value=double("nan");
     }
   Print("The second method to check and get calendar values");
   ArrayPrint(values_adjusted_2);

//--- option 3 to get the values - without checks
   AdjustedCalendarValue values_adjusted_3[];
   ArrayResize(values_adjusted_3, total);
//--- copy the values with checks and adjustments
   for(int i=0; i<total; i++)
     {
      values_adjusted_3[i].id=values[i].id;
      values_adjusted_3[i].event_id=values[i].event_id;
      values_adjusted_3[i].time=values[i].time;
      values_adjusted_3[i].period=values[i].period;
      values_adjusted_3[i].revision=values[i].revision;
      values_adjusted_3[i].impact_type=values[i].impact_type;
      //--- get values without checks
      values_adjusted_3[i].actual_value=values[i].GetActualValue();
      values_adjusted_3[i].prev_value=values[i].GetPreviousValue();
      values_adjusted_3[i].revised_prev_value=values[i].GetRevisedValue();
      values_adjusted_3[i].forecast_value=values[i].GetForecastValue();
     }
   Print("The third method to get calendar values - without checks");
   ArrayPrint(values_adjusted_3);
  }
/*
   We have received event values for country_code=EU: 1051
  Output the calendar values as they are
         [id] [event_id]              [time]            [period] [revision]       [act
   [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3
   [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0
   [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0 -92233720
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0
  The first method to check and get calendar values
         [id] [event_id]              [time]            [period] [revision] [actual_v
   [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.
   [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.
   [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.
  The second method to check and get calendar values
```

```
         [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.2
   [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.1
   [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.0
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.1
  The third method to get calendar values - without checks
         [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 144520  999500001 2021.01.04 12:00:00 2020.12.01 00:00:00          3       55.2
   [1] 144338  999520001 2021.01.04 23:30:00 2020.12.29 00:00:00          0      143.1
   [2] 147462  999010020 2021.01.04 23:45:00 1970.01.01 00:00:00          0
   [3] 111618  999010018 2021.01.05 12:00:00 2020.11.01 00:00:00          0       11.0
   [4] 111619  999010019 2021.01.05 12:00:00 2020.11.01 00:00:00          0        3.1
*/
```

Event frequency is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_FREQUENCY**

ID	Description
CALENDAR_FREQUENCY_NONE	Release frequency is not set
CALENDAR_FREQUENCY_WEEK	Released once a week
CALENDAR_FREQUENCY_MONTH	Released once a month
CALENDAR_FREQUENCY_QUARTER	Released once a quarter
CALENDAR_FREQUENCY_YEAR	Released once a year
CALENDAR_FREQUENCY_DAY	Released once a day

Event type is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_TYPE**

ID	Description
CALENDAR_TYPE_EVENT	Event (meeting, speech, etc.)
CALENDAR_TYPE_INDICATOR	Indicator
CALENDAR_TYPE_HOLIDAY	Holiday

A sector of the economy an event is related to is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_SECTOR**

ID	Description
CALENDAR_SECTOR_NONE	Sector is not set

ID	Description
CALENDAR_SECTOR_MARKET	Market, exchange
CALENDAR_SECTOR_GDP	Gross Domestic Product (GDP)
CALENDAR_SECTOR_JOBS	Labor market
CALENDAR_SECTOR_PRICES	Prices
CALENDAR_SECTOR_MONEY	Money
CALENDAR_SECTOR_TRADE	Trading
CALENDAR_SECTOR_GOVERNMENT	Government
CALENDAR_SECTOR_BUSINESS	Business
CALENDAR_SECTOR_CONSUMER	Consumption
CALENDAR_SECTOR_HOUSING	Housing
CALENDAR_SECTOR_TAXES	Taxes
CALENDAR_SECTOR_HOLIDAYS	Holidays

Event importance is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_IMPORTANCE**

ID	Description
CALENDAR_IMPORTANCE_NONE	Importance is not set
CALENDAR_IMPORTANCE_LOW	Low importance
CALENDAR_IMPORTANCE_MODERATE	Medium importance
CALENDAR_IMPORTANCE_HIGH	High importance

Measurement unit type used in displaying event values is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_UNIT**

ID	Description
CALENDAR_UNIT_NONE	Measurement unit is not set
CALENDAR_UNIT_PERCENT	Percentage
CALENDAR_UNIT_CURRENCY	National currency
CALENDAR_UNIT_HOUR	Hours
CALENDAR_UNIT_JOB	Jobs
CALENDAR_UNIT_RIG	Drilling rigs

ID	Description
CALENDAR_UNIT_USD	USD
CALENDAR_UNIT_PEOPLE	People
CALENDAR_UNIT_MORTGAGE	Mortgage loans
CALENDAR_UNIT_VOTE	Votes
CALENDAR_UNIT_BARREL	Barrels
CALENDAR_UNIT_CUBICFEET	Cubic feet
CALENDAR_UNIT_POSITION	Non-commercial net positions
CALENDAR_UNIT_BUILDING	Buildings

In some cases, economic parameter values require a multiplier set in the MqlCalendarEvent structure. Possible multiplier values are set in the listing **ENUM_CALENDAR_EVENT_MULTIPLIER**

ID	Description
CALENDAR_MULTIPLIER_NONE	Multiplier is not set
CALENDAR_MULTIPLIER_THOUSANDS	Thousands
CALENDAR_MULTIPLIER_MILLIONS	Millions
CALENDAR_MULTIPLIER_BILLIONS	Billions
CALENDAR_MULTIPLIER_TRILLIONS	Trillions

Event's potential impact on a national currency rate is indicated in the MqlCalendarValue structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_IMPACT**

ID	Description
CALENDAR_IMPACT_NA	Impact is not set
CALENDAR_IMPACT_POSITIVE	Positive impact
CALENDAR_IMPACT_NEGATIVE	Negative impact

Event time is specified in the MqlCalendarEvent structure. Possible values are set in the listing **ENUM_CALENDAR_EVENT_TIMEMODE**

ID	Description
CALENDAR_TIMEMODE_DATETIME	Source publishes an exact time of an event
CALENDAR_TIMEMODE_DATE	Event takes all day

ID	Description
CALENDAR_TIMEMODE_NOTIME	Source publishes no time of an event
CALENDAR_TIMEMODE_TENTATIVE	Source publishes a day of an event rather than its exact time. The time is specified upon the occurrence of the event.

See also

Economic Calendar

Codes of Errors and Warnings

This section contains the following descriptions:

- Return codes of the trade server – analyzing results of the trade request sent by function OrderSend();
- Compiler warnings – codes of warning messages that appear at compilation (not errors);
- Compilation errors – codes of error messages at an unsuccessful attempt to compile;
- Runtime errors – error codes in the execution of mql5-programs, which can be obtained using the GetLastError() function.

Return Codes of the Trade Server

All requests to execute trade operations are sent as a structure of a trade request MqlTradeRequest using function OrderSend(). The function execution result is placed to structure MqlTradeResult, whose *retcode* field contains the trade server return code.

Code	Constant	Description
10004	TRADE_RETCODE_REQUOTE	Requote
10006	TRADE_RETCODE_REJECT	Request rejected
10007	TRADE_RETCODE_CANCEL	Request canceled by trader
10008	TRADE_RETCODE_PLACED	Order placed
10009	TRADE_RETCODE_DONE	Request completed
10010	TRADE_RETCODE_DONE_PARTIAL	Only part of the request was completed
10011	TRADE_RETCODE_ERROR	Request processing error
10012	TRADE_RETCODE_TIMEOUT	Request canceled by timeout
10013	TRADE_RETCODE_INVALID	Invalid request
10014	TRADE_RETCODE_INVALID_VOLUME	Invalid volume in the request
10015	TRADE_RETCODE_INVALID_PRICE	Invalid price in the request
10016	TRADE_RETCODE_INVALID_STOPS	Invalid stops in the request
10017	TRADE_RETCODE_TRADE_DISABLED	Trade is disabled
10018	TRADE_RETCODE_MARKET_CLOSED	Market is closed
10019	TRADE_RETCODE_NO_MONEY	There is not enough money to complete the request
10020	TRADE_RETCODE_PRICE_CHANGED	Prices changed
10021	TRADE_RETCODE_PRICE_OFF	There are no quotes to process the request
10022	TRADE_RETCODE_INVALID_EXPIRATION	Invalid order expiration date in the request
10023	TRADE_RETCODE_ORDER_CHANGED	Order state changed
10024	TRADE_RETCODE_TOO_MANY_REQUESTS	Too frequent requests
10025	TRADE_RETCODE_NO_CHANGES	No changes in request
10026	TRADE_RETCODE_SERVER_DISABLES_AT	Autotrading disabled by server
10027	TRADE_RETCODE_CLIENT_DISABLES_AT	Autotrading disabled by client terminal
10028	TRADE_RETCODE_LOCKED	Request locked for processing
10029	TRADE_RETCODE_FROZEN	Order or position frozen
10030	TRADE_RETCODE_INVALID_FILL	Invalid order filling type

Code	Constant	Description
10031	TRADE_RETCODE_CONNECTION	No connection with the trade server
10032	TRADE_RETCODE_ONLY_REAL	Operation is allowed only for live accounts
10033	TRADE_RETCODE_LIMIT_ORDERS	The number of pending orders has reached the limit
10034	TRADE_RETCODE_LIMIT_VOLUME	The volume of orders and positions for the symbol has reached the limit
10035	TRADE_RETCODE_INVALID_ORDER	Incorrect or prohibited order type
10036	TRADE_RETCODE_POSITION_CLOSED	Position with the specified POSITION_IDENTIFIER has already been closed
10038	TRADE_RETCODE_INVALID_CLOSE_VOLUME	A close volume exceeds the current position volume
10039	TRADE_RETCODE_CLOSE_ORDER_EXIST	A close order already exists for a specified position. This may happen when working in the hedging system: • when attempting to close a position with an opposite one, while close orders for the position already exist • when attempting to fully or partially close a position if the total volume of the already present close orders and the newly placed one exceeds the current position volume
10040	TRADE_RETCODE_LIMIT_POSITIONS	The number of open positions simultaneously present on an account can be limited by the server settings. After a limit is reached, the server returns the TRADE_RETCODE_LIMIT_POSITIONS error when attempting to place an order. The limitation operates differently depending on the position accounting type: • Netting — number of open positions is considered. When a limit is reached, the platform does not let placing new orders whose execution may increase the number of open positions. In fact, the platform allows placing orders only for the symbols that already have open positions. The current pending orders are not considered since their execution may lead to changes in the current positions but it cannot increase their number.

Code	Constant	Description
		• Hedging — pending orders are considered together with open positions, since a pending order activation always leads to opening a new position. When a limit is reached, the platform does not allow placing both new market orders for opening positions and pending orders.
10041	TRADE_RETCODE_REJECT_CANCEL	The pending order activation request is rejected, the order is canceled
10042	TRADE_RETCODE_LONG_ONLY	The request is rejected, because the "**Only long positions are allowed**" rule is set for the symbol (POSITION_TYPE_BUY)
10043	TRADE_RETCODE_SHORT_ONLY	The request is rejected, because the "**Only short positions are allowed**" rule is set for the symbol (POSITION_TYPE_SELL)
10044	TRADE_RETCODE_CLOSE_ONLY	The request is rejected, because the "**Only position closing is allowed**" rule is set for the symbol
10045	TRADE_RETCODE_FIFO_CLOSE	The request is rejected, because "**Position closing is allowed only by FIFO rule**" flag is set for the trading account (ACCOUNT_FIFO_CLOSE=true)
10046	TRADE_RETCODE_HEDGE_PROHIBITED	The request is rejected, because the "**Opposite positions on a single symbol are disabled**" rule is set for the trading account. For example, if the account has a Buy position, then a user cannot open a Sell position or place a pending sell order. The rule is only applied to accounts with hedging accounting system (ACCOUNT_MARGIN_MODE=ACCOUNT_MARGIN_MODE_RETAIL_HEDGING).

Compiler Warnings

Compiler warnings are shown for informational purposes only and are not error messages.

Code	Description
21	Incomplete record of a date in the datetime string
22	Wrong number in the datetime string for the date. Requirements: Year 1970 <= X <= 3000 Month 0 <X <= 12 Day 0 <X <= 31/30/28 (29)....
23	Wrong number of datetime string for time. Requirements: Hour 0 <= X <24 Minute 0 <= X <60
24	Invalid color in RGB format: one of RGB components is less than 0 or greater than 255
25	Unknown character of the escape sequences. Known: \n \r \t \\ \" \' \X \x
26	Too large volume of local variables (> 512Kb) of the function, reduce the number
29	Enumeration already defined (duplication) - members will be added to the first definition
30	Overriding macro
31	The variable is declared but is not used anywhere
32	Constructor must be of void type
33	Destructor must be of void type
34	Constant does not fit in the range of integers (X> _UI64_MAX | | X <_I64_MIN) and will be converted to the double type
35	Too long HEX - more than 16 significant characters (senior nibbles are cut)
36	No nibbles in HEX string "0x"
37	No function - nothing to be performed
38	A non-initialized variable is used
41	Function has no body, and is not called
43	Possible loss of data at typecasting. Example: int x = (double) z;
44	Loss of accuracy (of data) when converting a constant. Example: int x = M_PI
45	Difference between the signs of operands in the operations of comparison. Example: (char) c1> (uchar) c2
46	Problems with function importing - declaration of #import is required or import of functions is closed

Code	Description
47	Too large description - extra characters will not be included in the executable file
48	The number of indicator buffers declared is less than required
49	No color to plot a graphical series in the indicator
50	No graphical series to draw the indicator
51	'OnStart' handler function not found in the script
52	'OnStart' handler function is defined with wrong parameters
53	'OnStart' function can be defined only in a script
54	'OnInit' function is defined with wrong parameters
55	'OnInit' function is not used in scripts
56	'OnDeinit' function is defined with wrong parameters
57	'OnDeinit' function is not used in scripts
58	Two 'OnCalculate' functions are defined. OnCalculate () at one price array will be used
59	Overfilling detected when calculating a complex integer constant
60	Probably, the variable is not initialized.
61	This declaration makes it impossible to refer to the local variable declared on the specified line
62	This declaration makes it impossible to refer to the global variable declared on the specified line
63	Cannot be used for static allocated array
64	This variable declaration hides predefined variable
65	The value of the expression is always true/false
66	Using a variable or bool type expression in mathematical operations is unsafe
67	The result of applying the unary minus operator to an unsigned ulong type is undefined
68	The version specified in the #property version property is unacceptable for the Market section; the correct format of #property version id "XXX.YYY"
69	Empty controlled statement found
70	Invalid function return type or incorrect parameters during declaration of the event handler function
71	An implicit cast of structures to one type is required
72	This declaration makes direct access to the member of a class declared in the specified string impossible. Access will be possible only with the scope resolution operation ::

Code	Description
73	Binary constant is too big, high-order digits will be truncated
74	Parameter in the method of the inherited class has a different const modifier, the derived function has overloaded the parent function
75	Negative or too large shift value in shift bitwise operation, execution result is undefined
76	Function must return a value
77	void function returns a value
78	Not all control paths return a value
79	Expressions are not allowed on a global scope
80	Check operator precedence for possible error; use parentheses to clarify precedence
81	Two OnCalCulate() are defined. OHLC version will be used
82	Struct has no members, size assigned to 1 byte
83	Return value of the function should be checked
84	Resource indicator is compiled for debugging. That slows down the performance. Please recompile the indicator to increase performance
85	Too great character code in the string, must be in the range 0 to 65535
86	Unrecognized character in the string
87	No indicator window property (setting the display in the main window or a subwindow) is defined. Property #property indicator_chart_window is applied

Compilation Errors

MetaEdtior 5 shows error messages about the program errors detected by the built-in compiler during compilation. The list of these errors is given below in table. To compile a source code into an executable one, press **F7**. Programs that contain errors cannot be compiled until the errors identified by the compiler are eliminated.

Code	Description
100	File reading error
101	Error of opening an *. EX5 for writing
103	Not enough free memory to complete compilation
104	Empty syntactic unit unrecognized by compiler
105	Incorrect file name in #include
106	Error accessing a file in #include (probably the file does not exist)
108	Inappropriate name for #define
109	Unknown command of preprocessor (valid #include, #define, #property, #import)
110	Symbol unknown to compiler
111	Function not implemented (description is present, but no body)
112	Double quote (") omitted
113	Opening angle bracket (<) or double quote (") omitted
114	Single quote (') omitted
115	Closing angle bracket ">" omitted
116	Type not specified in declaration
117	No return operator or return is found not in all branches of the implementation
118	Opening bracket of call parameters was expected
119	Error writing EX5
120	Invalid access to an array
121	The function is not of void type and the return operator must return a value
122	Incorrect declaration of the destructor
123	Colon ":" is missing
124	Variable is already declared
125	Variable with such identifier already declared
126	Variable name is too long (> 250 characters)
127	Structure with such identifier already defined

Code	Description
128	Structure is not defined
129	Structure member with the same name already defined
130	No such structure member
131	Breached pairing of brackets
132	Opening parenthesis "(" expected
133	Unbalanced braces (no "}")
134	Difficult to compile (too much branching, internal stack levels are overfilled)
135	Error of file opening for reading
136	Not enough memory to download the source file into memory
137	Variable is expected
138	Reference cannot be initialized
140	Assignment expected (appears at declaration)
141	Opening brace "{" expected
142	Parameter can be a dynamic array only
143	Use of "void" type is unacceptable
144	No pair for ")" or "]", i.e. "(or" [" is absent
145	No pair for "(or" [", i.e. ") "or"] " is absent
146	Incorrect array size
147	Too many parameters (> 64)
149	This token is not expected here
150	Invalid use of operation (invalid operands)
151	Expression of void type not allowed
152	Operator is expected
153	Misuse of break
154	Semicolon ";" expected
155	Comma "," expected
156	Must be a class type, not struct
157	Expression is expected
158	"non HEX character" found in HEX or too long number (number of digits> 511)
159	String-constant has more than 65534 characters

Code	Description
160	Function definition is unacceptable here
161	Unexpected end of program
162	Forward declaration is prohibited for structures
163	Function with this name is already defined and has another return type
164	Function with this name is already defined and has a different set of parameters
165	Function with this name is already defined and implemented
166	Function overload for this call was not found
167	Function with a return value of void type cannot return a value
168	Function is not defined
170	Value is expected
171	In *case* expression only integer constants are valid
172	The value of *case* in this *switch* is already used
173	Integer is expected
174	In #import expression file name is expected
175	Expressions are not allowed on global level
176	Omitted parenthesis ")" before ";"
177	To the left of equality sign a variable is expected
178	The result of expression is not used
179	Declaring of variables is not allowed in *case*
180	Implicit conversion from a string to a number
181	Implicit conversion of a number to a string
182	Ambiguous call of an overloaded function (several overloads fit)
183	Illegal *else* without proper *if*
184	Invalid *case* or *default* without a *switch*
185	Inappropriate use of ellipsis
186	The initializing sequence has more elements than the initialized variable
187	A constant for *case* expected
188	A constant expression required
189	A constant variable cannot be changed
190	Closing bracket or a comma is expected (declaring array member)

Code	Description
191	Enumerator identifier already defined
192	Enumeration cannot have access modifiers (const, extern, static)
193	Enumeration member already declared with a different value
194	There is a variable defined with the same name
195	There is a structure defined with the same name
196	Name of enumeration member expected
197	Integer expression expected
198	Division by zero in constant expression
199	Wrong number of parameters in the function
200	Parameter by reference must be a variable
201	Variable of the same type to pass by reference expected
202	A constant variable cannot be passed by a non-constant reference
203	Requires a positive integer constant
204	Failed to access protected class member
205	Import already defined in another way
208	Executable file not created
209	'OnCalculate' entry point not found for the indicator
210	The continue operation can be used only inside a loop
211	Error accessing private (closed) class member
213	Method of structure or class is not declared
214	Error accessing private (closed) class method
216	Copying of structures with objects is not allowed
218	Index out of array range
219	Array initialization in structure or class declaration not allowed
220	Class constructor cannot have parameters
221	Class destructor can not have parameters
222	Class method or structure with the same name and parameters have already been declared
223	Operand expected
224	Class method or structure with the same name exists, but with different parameters (declaration!=implementation)

Code	Description
225	Imported function is not described
226	ZeroMemory() is not allowed for objects with protected members or inheritance
227	Ambiguous call of the overloaded function (exact match of parameters for several overloads)
228	Variable name expected
229	A reference cannot be declared in this place
230	Already used as the enumeration name
232	Class or structure expected
235	Cannot call 'delete' operator to delete the array
236	Operator ' while' expected
237	Operator 'delete' must have a pointer
238	There is 'default' for this 'switch' already
239	Syntax error
240	Escape-sequence can occur only in strings (starts with '\')
241	Array required - square bracket '[' does not apply to an array, or non arrays are passed as array parameters
242	Can not be initialized through the initialization sequence
243	Import is not defined
244	Optimizer error on the syntactic tree
245	Declared too many structures (try to simplify the program)
246	Conversion of the parameter is not allowed
247	Incorrect use of the 'delete' operator
248	It's not allowed to declare a pointer to a reference
249	It's not allowed to declare a reference to a reference
250	It's not allowed to declare a pointer to a pointer
251	Structure declaration in the list of parameter is not allowed
252	Invalid operation of typecasting
253	A pointer can be declared only for a class or structure
256	Undeclared identifier
257	Executable code optimizer error
258	Executable code generation error

Code	Description
260	Invalid expression for the 'switch' operator
261	Pool of string constants overfilled, simplify program
262	Cannot convert to enumeration
263	Do not use 'virtual' for data (members of a class or structure)
264	Cannot call protected method of class
265	Overridden virtual functions return a different type
266	Class cannot be inherited from a structure
267	Structure cannot be inherited from a class
268	Constructor cannot be virtual (*virtual* specifier is not allowed)
269	Method of structure cannot be virtual
270	Function must have a body
271	Overloading of system functions (terminal functions) is prohibited
272	*Const* specifier is invalid for functions that are not members of a class or structure
274	Not allowed to change class members in constant method
276	Inappropriate initialization sequence
277	Missed default value for the parameter (specific declaration of default parameters)
278	Overriding the default parameter (different values in declaration and implementation)
279	Not allowed to call non-constant method for a constant object
280	An object is necessary for accessing members (a dot for a non class/structure is specified)
281	The name of an already declared structure cannot be used in declaration
284	Unauthorized conversion (at closed inheritance)
285	Structures and arrays cannot be used as input variables
286	*Const* specifier is not valid for constructor/destructor
287	Incorrect string expression for a datetime
288	Unknown property (#property)
289	Incorrect value of a property
290	Invalid index for a property in #property
291	Call parameter omitted - <func (x,)>
293	Object must be passed by reference

Code	Description
294	Array must be passed by reference
295	Function was declared as exportable
296	Function was not declared as exportable
297	It is prohibited to export imported function
298	Imported function cannot have this parameter (prohibited to pass a pointer, class or structure containing a dynamic array, pointer, class, etc.)
299	Must be a class
300	#import was not closed
302	Type mismatch
303	Extern variable is already initialized
304	No exported function or entry point found
305	Explicit constructor call is not allowed
306	Method was declared as constant
307	Method was not declared as constant
308	Incorrect size of the resource file
309	Incorrect resource name
310	Resource file opening error
311	Resource file reading error
312	Unknown resource type
313	Incorrect path to the resource file
314	The specified resource name is already used
315	Argument expected for the function-like macro
316	Unexpected symbol in macro definition
317	Error in formal parameters of the macro
318	Invalid number of parameters for a macro
319	Too many parameters for a macro
320	Too complex, simplify the macro
321	Parameter for EnumToString() can be only an enumeration
322	The resource name is too long
323	Unsupported image format (only BMP with 24 or 32 bit color depth is supported)
324	An array cannot be declared in operator

Code	Description
325	The function can be declared only in the global scope
326	The declaration is not allowed for the current scope
327	Initialization of static variables with the values of local variables is not allowed
328	Illegal declaration of an array of objects that do not have a default constructor
329	Initialization list allowed only for constructors
330	No function definition after initialization list
331	Initialization list is empty
332	Array initialization in a constructor is not allowed
333	Initializing members of a parent class in the initialization list is not allowed
334	Expression of the integer type expected
335	Memory required for the array exceeds the maximum value
336	Memory required for the structure exceeds the maximum value
337	Memory required for the variables declared on the global level exceeds the maximum value
338	Memory required for local variables exceeds the maximum value
339	Constructor not defined
340	Invalid name of the icon file
341	Could not open the icon file at the specified path
342	The icon file is incorrect and is not of the ICO format
343	Reinitialization of a member in a class/structure constructor using the initialization list
344	Initialization of static members in the constructor initialization list is not allowed
345	Initialization of a non-static member of a class/structure on a global level is not allowed
346	The name of the class/structure method matches the name of an earlier declared member
347	The name of the class/structure member matches the name of an earlier declared method
348	Virtual function cannot be declared as static
349	The const modifier is not allowed for static functions
350	Constructor or destructor cannot be static
351	Non-static member/method of a class or a structure cannot be accessed from a static function

Code	Description
352	An overload operation (+,-,[],++,-- etc.) is expected after the operator keyword
353	Not all operations can be overloaded in MQL5
354	Definition does not match declaration
355	An invalid number of parameters is specified for the operator
356	Event handling function not found
357	Method cannot be exported
358	A pointer to the constant object cannot be normalized by a non-constant object
359	Class templates are not supported yet
360	Function template overload is not supported yet
361	Function template cannot be applied
362	Ambiguous parameter in function template (several parameter types can be applied)
363	Unable to determine the parameter type, by which the function template argument should be normalized
364	Incorrect number of parameters in the function template
365	Function template cannot be virtual
366	Function templates cannot be exported
367	Function templates cannot be imported
368	Structures containing the objects are not allowed
369	String arrays and structures containing the objects are not allowed
370	A static class/structure member must be explicitly initialized
371	Compiler limitation: the string cannot contain more than 65 535 characters
372	Inconsistent #ifdef/#endif
373	Object of class cannot be returned, copy constructor not found
374	Non-static members and methods cannot be used
375	OnTesterInit() impossible to use without OnTesterDeinit()
376	Redefinition of formal parameter '%s'
377	Macro __FUNCSIG__ and __FUNCTION__ cannot appear outside of a function body
378	Invalid returned type. For example, this error will be produced for functions imported from DLL that return structure or pointer.
379	Template usage error
380	Not used

Code	Description
381	Illegal syntax when declaring pure virtual function, only "=NULL" or "=0" are allowed
382	Only virtual functions can be declared with the pure-specifier ("=NULL" or "=0")
383	Abstract class cannot be instantiated
384	A pointer to a user-defined type should be applied as a target type for dynamic casting using the dynamic_cast operator
385	"Pointer to function" type is expected
386	Pointers to methods are not supported
387	Error – cannot define the type of a pointer to function
388	Type cast is not available due to private inheritance
389	A variable with const modifier should be initialized during declaration
393	Only methods with public access can be declared in an interface
394	Invalid nesting of an interface inside of another interface
395	An interface can only be derived from another interface
396	An interface is expected
397	Interfaces only support public inheritance
398	An interface cannot contain members
399	Interface objects cannot be created directly, only use inheritance
400	A specifier cannot be used in a forward declaration
401	Inheritance from the class is impossible, since it is declared with the final specifier
402	Cannot redefine a method declared with the final specifier
403	The final specifier can be applied only to virtual functions
404	The method marked by the override specifier actually does not override any base class function
405	A specifier is not allowed in defining a function, but only in declaring
406	Cannot cast the type to the specified one
407	The type cannot be used for a resource variable
408	Error in the project file
409	Cannot be used as a union member
410	Ambiguous choice for the name, the usage context should be explicitly defined
411	The structure cannot be used from DLL
412	Cannot call a function marked by the delete specifier

Code	Description
413	MQL4 is not supported. To compile this program, use MetaEditor from your MetaTrader 4 installation folder

Runtime Errors

GetLastError() is the function that returns the last error code that is stored in the predefined variable _LastError. This value can be reset to zero by the ResetLastError() function.

Constant	Code	Description
ERR_SUCCESS	0	The operation completed successfully
ERR_INTERNAL_ERROR	4001	Unexpected internal error
ERR_WRONG_INTERNAL_PARAMETER	4002	Wrong parameter in the inner call of the client terminal function
ERR_INVALID_PARAMETER	4003	Wrong parameter when calling the system function
ERR_NOT_ENOUGH_MEMORY	4004	Not enough memory to perform the system function
ERR_STRUCT_WITHOBJECTS_ORCLASS	4005	The structure contains objects of strings and/or dynamic arrays and/or structure of such objects and/or classes
ERR_INVALID_ARRAY	4006	Array of a wrong type, wrong size, or a damaged object of a dynamic array
ERR_ARRAY_RESIZE_ERROR	4007	Not enough memory for the relocation of an array, or an attempt to change the size of a static array
ERR_STRING_RESIZE_ERROR	4008	Not enough memory for the relocation of string
ERR_NOTINITIALIZED_STRING	4009	Not initialized string
ERR_INVALID_DATETIME	4010	Invalid date and/or time
ERR_ARRAY_BAD_SIZE	4011	Total amount of elements in the array cannot exceed 2147483647
ERR_INVALID_POINTER	4012	Wrong pointer
ERR_INVALID_POINTER_TYPE	4013	Wrong type of pointer
ERR_FUNCTION_NOT_ALLOWED	4014	Function is not allowed for call
ERR_RESOURCE_NAME_DUPLICATED	4015	The names of the dynamic and the static resource match
ERR_RESOURCE_NOT_FOUND	4016	Resource with this name has not been found in EX5

Constant	Code	Description
ERR_RESOURCE_UNSUPPORTED_TYPE	4017	Unsupported resource type or its size exceeds 16 Mb
ERR_RESOURCE_NAME_IS_TOO_LONG	4018	The resource name exceeds 63 characters
ERR_MATH_OVERFLOW	4019	Overflow occurred when calculating math function
ERR_SLEEP_ERROR	4020	Out of test end date after calling Sleep()
ERR_PROGRAM_STOPPED	4022	Test forcibly stopped from the outside. For example, optimization interrupted, visual testing window closed or testing agent stopped
ERR_INVALID_TYPE	4023	Invalid type
ERR_INVALID_HANDLE	4024	Invalid handle
ERR_TOO_MANY_OBJECTS	4025	Object pool filled out
Charts		
ERR_CHART_WRONG_ID	4101	Wrong chart ID
ERR_CHART_NO_REPLY	4102	Chart does not respond
ERR_CHART_NOT_FOUND	4103	Chart not found
ERR_CHART_NO_EXPERT	4104	No Expert Advisor in the chart that could handle the event
ERR_CHART_CANNOT_OPEN	4105	Chart opening error
ERR_CHART_CANNOT_CHANGE	4106	Failed to change chart symbol and period
ERR_CHART_WRONG_PARAMETER	4107	Error value of the parameter for the function of working with charts
ERR_CHART_CANNOT_CREATE_TIMER	4108	Failed to create timer
ERR_CHART_WRONG_PROPERTY	4109	Wrong chart property ID
ERR_CHART_SCREENSHOT_FAILED	4110	Error creating screenshots
ERR_CHART_NAVIGATE_FAILED	4111	Error navigating through chart
ERR_CHART_TEMPLATE_FAILED	4112	Error applying template
ERR_CHART_WINDOW_NOT_FOUND	4113	Subwindow containing the indicator was not found

Constant	Code	Description
ERR_CHART_INDICATOR_CANNOT_ADD	4114	Error adding an indicator to chart
ERR_CHART_INDICATOR_CANNOT_DEL	4115	Error deleting an indicator from the chart
ERR_CHART_INDICATOR_NOT_FOUND	4116	Indicator not found on the specified chart
Graphical Objects		
ERR_OBJECT_ERROR	4201	Error working with a graphical object
ERR_OBJECT_NOT_FOUND	4202	Graphical object was not found
ERR_OBJECT_WRONG_PROPERTY	4203	Wrong ID of a graphical object property
ERR_OBJECT_GETDATE_FAILED	4204	Unable to get date corresponding to the value
ERR_OBJECT_GETVALUE_FAILED	4205	Unable to get value corresponding to the date
MarketInfo		
ERR_MARKET_UNKNOWN_SYMBOL	4301	Unknown symbol
ERR_MARKET_NOT_SELECTED	4302	Symbol is not selected in MarketWatch
ERR_MARKET_WRONG_PROPERTY	4303	Wrong identifier of a symbol property
ERR_MARKET_LASTTIME_UNKNOWN	4304	Time of the last tick is not known (no ticks)
ERR_MARKET_SELECT_ERROR	4305	Error adding or deleting a symbol in MarketWatch
ERR_MARKET_SELECT_LIMIT	4306	Exceeded the limit of selected symbols in MarketWatch
ERR_MARKET_SESSION_INDEX	4307	Wrong session ID when calling the SymbolInfoSessionQuote/SymbolInfoSessionTrade function
History Access		
ERR_HISTORY_NOT_FOUND	4401	Requested history not found
ERR_HISTORY_WRONG_PROPERTY	4402	Wrong ID of the history property
ERR_HISTORY_TIMEOUT	4403	Exceeded history request timeout
ERR_HISTORY_BARS_LIMIT	4404	Number of requested bars limited by terminal settings

Constant	Code	Description
ERR_HISTORY_LOAD_ERRORS	4405	Multiple errors when loading history
ERR_HISTORY_SMALL_BUFFER	4407	Receiving array is too small to store all requested data
Global_Variables		
ERR_GLOBALVARIABLE_NOT_FOUND	4501	Global variable of the client terminal is not found
ERR_GLOBALVARIABLE_EXISTS	4502	Global variable of the client terminal with the same name already exists
ERR_GLOBALVARIABLE_NOT_MODIFIED	4503	Global variables were not modified
ERR_GLOBALVARIABLE_CANNOTREAD	4504	Cannot read file with global variable values
ERR_GLOBALVARIABLE_CANNOTWRITE	4505	Cannot write file with global variable values
ERR_MAIL_SEND_FAILED	4510	Email sending failed
ERR_PLAY_SOUND_FAILED	4511	Sound playing failed
ERR_MQL5_WRONG_PROPERTY	4512	Wrong identifier of the program property
ERR_TERMINAL_WRONG_PROPERTY	4513	Wrong identifier of the terminal property
ERR_FTP_SEND_FAILED	4514	File sending via ftp failed
ERR_NOTIFICATION_SEND_FAILED	4515	Failed to send a notification
ERR_NOTIFICATION_WRONG_PARAMETER	4516	Invalid parameter for sending a notification – an empty string or NULL has been passed to the SendNotification() function
ERR_NOTIFICATION_WRONG_SETTINGS	4517	Wrong settings of notifications in the terminal (ID is not specified or permission is not set)
ERR_NOTIFICATION_TOO_FREQUENT	4518	Too frequent sending of notifications
ERR_FTP_NOSERVER	4519	FTP server is not specified
ERR_FTP_NOLOGIN	4520	FTP login is not specified
ERR_FTP_FILE_ERROR	4521	File not found in the MQL5\Files directory to send on FTP server

Constant	Code	Description
ERR_FTP_CONNECT_FAILED	4522	FTP connection failed
ERR_FTP_CHANGEDIR	4523	FTP path not found on server
Custom Indicator Buffers		
ERR_BUFFERS_NO_MEMORY	4601	Not enough memory for the distribution of indicator buffers
ERR_BUFFERS_WRONG_INDEX	4602	Wrong indicator buffer index
Custom Indicator Properties		
ERR_CUSTOM_WRONG_PROPERTY	4603	Wrong ID of the custom indicator property
Account		
ERR_ACCOUNT_WRONG_PROPERTY	4701	Wrong account property ID
ERR_TRADE_WRONG_PROPERTY	4751	Wrong trade property ID
ERR_TRADE_DISABLED	4752	Trading by Expert Advisors prohibited
ERR_TRADE_POSITION_NOT_FOUND	4753	Position not found
ERR_TRADE_ORDER_NOT_FOUND	4754	Order not found
ERR_TRADE_DEAL_NOT_FOUND	4755	Deal not found
ERR_TRADE_SEND_FAILED	4756	Trade request sending failed
ERR_TRADE_CALC_FAILED	4758	Failed to calculate profit or margin
Indicators		
ERR_INDICATOR_UNKNOWN_SYMBOL	4801	Unknown symbol
ERR_INDICATOR_CANNOT_CREATE	4802	Indicator cannot be created
ERR_INDICATOR_NO_MEMORY	4803	Not enough memory to add the indicator
ERR_INDICATOR_CANNOT_APPLY	4804	The indicator cannot be applied to another indicator
ERR_INDICATOR_CANNOT_ADD	4805	Error applying an indicator to chart
ERR_INDICATOR_DATA_NOT_FOUND	4806	Requested data not found
ERR_INDICATOR_WRONG_HANDLE	4807	Wrong indicator handle
ERR_INDICATOR_WRONG_PARAMETERS	4808	Wrong number of parameters when creating an indicator

Constant	Code	Description
ERR_INDICATOR_PARAMETERS_MISSING	4809	No parameters when creating an indicator
ERR_INDICATOR_CUSTOM_NAME	4810	The first parameter in the array must be the name of the custom indicator
ERR_INDICATOR_PARAMETER_TYPE	4811	Invalid parameter type in the array when creating an indicator
ERR_INDICATOR_WRONG_INDEX	4812	Wrong index of the requested indicator buffer
Depth of Market		
ERR_BOOKS_CANNOT_ADD	4901	Depth Of Market can not be added
ERR_BOOKS_CANNOT_DELETE	4902	Depth Of Market can not be removed
ERR_BOOKS_CANNOT_GET	4903	The data from Depth Of Market can not be obtained
ERR_BOOKS_CANNOT_SUBSCRIBE	4904	Error in subscribing to receive new data from Depth Of Market
File Operations		
ERR_TOO_MANY_FILES	5001	More than 64 files cannot be opened at the same time
ERR_WRONG_FILENAME	5002	Invalid file name
ERR_TOO_LONG_FILENAME	5003	Too long file name
ERR_CANNOT_OPEN_FILE	5004	File opening error
ERR_FILE_CACHEBUFFER_ERROR	5005	Not enough memory for cache to read
ERR_CANNOT_DELETE_FILE	5006	File deleting error
ERR_INVALID_FILEHANDLE	5007	A file with this handle was closed, or was not opening at all
ERR_WRONG_FILEHANDLE	5008	Wrong file handle
ERR_FILE_NOTTOWRITE	5009	The file must be opened for writing
ERR_FILE_NOTTOREAD	5010	The file must be opened for reading
ERR_FILE_NOTBIN	5011	The file must be opened as a binary one
ERR_FILE_NOTTXT	5012	The file must be opened as a text

Constant	Code	Description
ERR_FILE_NOTTXTORCSV	5013	The file must be opened as a text or CSV
ERR_FILE_NOTCSV	5014	The file must be opened as CSV
ERR_FILE_READERROR	5015	File reading error
ERR_FILE_BINSTRINGSIZE	5016	String size must be specified, because the file is opened as binary
ERR_INCOMPATIBLE_FILE	5017	A text file must be for string arrays, for other arrays - binary
ERR_FILE_IS_DIRECTORY	5018	This is not a file, this is a directory
ERR_FILE_NOT_EXIST	5019	File does not exist
ERR_FILE_CANNOT_REWRITE	5020	File can not be rewritten
ERR_WRONG_DIRECTORYNAME	5021	Wrong directory name
ERR_DIRECTORY_NOT_EXIST	5022	Directory does not exist
ERR_FILE_ISNOT_DIRECTORY	5023	This is a file, not a directory
ERR_CANNOT_DELETE_DIRECTORY	5024	The directory cannot be removed
ERR_CANNOT_CLEAN_DIRECTORY	5025	Failed to clear the directory (probably one or more files are blocked and removal operation failed)
ERR_FILE_WRITEERROR	5026	Failed to write a resource to a file
ERR_FILE_ENDOFFILE	5027	Unable to read the next piece of data from a CSV file (FileReadString, FileReadNumber, FileReadDatetime, FileReadBool), since the end of file is reached
String Casting		
ERR_NO_STRING_DATE	5030	No date in the string
ERR_WRONG_STRING_DATE	5031	Wrong date in the string
ERR_WRONG_STRING_TIME	5032	Wrong time in the string
ERR_STRING_TIME_ERROR	5033	Error converting string to date
ERR_STRING_OUT_OF_MEMORY	5034	Not enough memory for the string
ERR_STRING_SMALL_LEN	5035	The string length is less than expected

Constant	Code	Description
ERR_STRING_TOO_BIGNUMBER	5036	Too large number, more than ULONG_MAX
ERR_WRONG_FORMATSTRING	5037	Invalid format string
ERR_TOO_MANY_FORMATTERS	5038	Amount of format specifiers more than the parameters
ERR_TOO_MANY_PARAMETERS	5039	Amount of parameters more than the format specifiers
ERR_WRONG_STRING_PARAMETER	5040	Damaged parameter of string type
ERR_STRINGPOS_OUTOFRANGE	5041	Position outside the string
ERR_STRING_ZEROADDED	5042	0 added to the string end, a useless operation
ERR_STRING_UNKNOWNTYPE	5043	Unknown data type when converting to a string
ERR_WRONG_STRING_OBJECT	5044	Damaged string object
Operations with Arrays		
ERR_INCOMPATIBLE_ARRAYS	5050	Copying incompatible arrays. String array can be copied only to a string array, and a numeric array - in numeric array only
ERR_SMALL_ASSERIES_ARRAY	5051	The receiving array is declared as AS_SERIES, and it is of insufficient size
ERR_SMALL_ARRAY	5052	Too small array, the starting position is outside the array
ERR_ZEROSIZE_ARRAY	5053	An array of zero length
ERR_NUMBER_ARRAYS_ONLY	5054	Must be a numeric array
ERR_ONEDIM_ARRAYS_ONLY	5055	Must be a one-dimensional array
ERR_SERIES_ARRAY	5056	Timeseries cannot be used
ERR_DOUBLE_ARRAY_ONLY	5057	Must be an array of type double
ERR_FLOAT_ARRAY_ONLY	5058	Must be an array of type float
ERR_LONG_ARRAY_ONLY	5059	Must be an array of type long
ERR_INT_ARRAY_ONLY	5060	Must be an array of type int
ERR_SHORT_ARRAY_ONLY	5061	Must be an array of type short
ERR_CHAR_ARRAY_ONLY	5062	Must be an array of type char
ERR_STRING_ARRAY_ONLY	5063	String array only

Constant	Code	Description
Operations with OpenCL		
ERR_OPENCL_NOT_SUPPORTED	5100	OpenCL functions are not supported on this computer
ERR_OPENCL_INTERNAL	5101	Internal error occurred when running OpenCL
ERR_OPENCL_INVALID_HANDLE	5102	Invalid OpenCL handle
ERR_OPENCL_CONTEXT_CREATE	5103	Error creating the OpenCL context
ERR_OPENCL_QUEUE_CREATE	5104	Failed to create a run queue in OpenCL
ERR_OPENCL_PROGRAM_CREATE	5105	Error occurred when compiling an OpenCL program
ERR_OPENCL_TOO_LONG_KERNEL_NAME	5106	Too long kernel name (OpenCL kernel)
ERR_OPENCL_KERNEL_CREATE	5107	Error creating an OpenCL kernel
ERR_OPENCL_SET_KERNEL_PARAMETER	5108	Error occurred when setting parameters for the OpenCL kernel
ERR_OPENCL_EXECUTE	5109	OpenCL program runtime error
ERR_OPENCL_WRONG_BUFFER_SIZE	5110	Invalid size of the OpenCL buffer
ERR_OPENCL_WRONG_BUFFER_OFFSET	5111	Invalid offset in the OpenCL buffer
ERR_OPENCL_BUFFER_CREATE	5112	Failed to create an OpenCL buffer
ERR_OPENCL_TOO_MANY_OBJECTS	5113	Too many OpenCL objects
ERR_OPENCL_SELECTDEVICE	5114	OpenCL device selection error
Working with databases		
ERR_DATABASE_INTERNAL	5120	Internal database error
ERR_DATABASE_INVALID_HANDLE	5121	Invalid database handle
ERR_DATABASE_TOO_MANY_OBJECTS	5122	Exceeded the maximum acceptable number of Database objects
ERR_DATABASE_CONNECT	5123	Database connection error
ERR_DATABASE_EXECUTE	5124	Request execution error
ERR_DATABASE_PREPARE	5125	Request generation error
ERR_DATABASE_NO_MORE_DATA	5126	No more data to read

Constant	Code	Description
ERR_DATABASE_STEP	5127	Failed to move to the next request entry
ERR_DATABASE_NOT_READY	5128	Data for reading request results are not ready yet
ERR_DATABASE_BIND_PARAMETERS	5129	Failed to auto substitute parameters to an SQL request
Operations with WebRequest		
ERR_WEBREQUEST_INVALID_ADDRESS	5200	Invalid URL
ERR_WEBREQUEST_CONNECT_FAILED	5201	Failed to connect to specified URL
ERR_WEBREQUEST_TIMEOUT	5202	Timeout exceeded
ERR_WEBREQUEST_REQUEST_FAILED	5203	HTTP request failed
Operations with network (sockets)		
ERR_NETSOCKET_INVALIDHANDLE	5270	Invalid socket handle passed to function
ERR_NETSOCKET_TOO_MANY_OPENED	5271	Too many open sockets (max 128)
ERR_NETSOCKET_CANNOT_CONNECT	5272	Failed to connect to remote host
ERR_NETSOCKET_IO_ERROR	5273	Failed to send/receive data from socket
ERR_NETSOCKET_HANDSHAKE_FAILED	5274	Failed to establish secure connection (TLS Handshake)
ERR_NETSOCKET_NO_CERTIFICATE	5275	No data on certificate protecting the connection
Custom Symbols		
ERR_NOT_CUSTOM_SYMBOL	5300	A custom symbol must be specified
ERR_CUSTOM_SYMBOL_WRONG_NAME	5301	The name of the custom symbol is invalid. The symbol name can only contain Latin letters without punctuation, spaces or special characters (may only contain ".", "_", "&" and "#"). It is not recommended to use characters <, >, :, ", /,\, |, ?, *.
ERR_CUSTOM_SYMBOL_NAME_LONG	5302	The name of the custom symbol is too long. The length of the symbol name must not exceed 32 characters including the ending 0 character

Constant	Code	Description
ERR_CUSTOM_SYMBOL_PATH_LONG	5303	The path of the custom symbol is too long. The path length should not exceed 128 characters including "Custom\\", the symbol name, group separators and the ending 0
ERR_CUSTOM_SYMBOL_EXIST	5304	A custom symbol with the same name already exists
ERR_CUSTOM_SYMBOL_ERROR	5305	Error occurred while creating, deleting or changing the custom symbol
ERR_CUSTOM_SYMBOL_SELECTED	5306	You are trying to delete a custom symbol selected in Market Watch
ERR_CUSTOM_SYMBOL_PROPERTY_WRONG	5307	An invalid custom symbol property
ERR_CUSTOM_SYMBOL_PARAMETER_ERROR	5308	A wrong parameter while setting the property of a custom symbol
ERR_CUSTOM_SYMBOL_PARAMETER_LONG	5309	A too long string parameter while setting the property of a custom symbol
ERR_CUSTOM_TICKS_WRONG_ORDER	5310	Ticks in the array are not arranged in the order of time
Economic Calendar		
ERR_CALENDAR_MORE_DATA	5400	Array size is insufficient for receiving descriptions of all values
ERR_CALENDAR_TIMEOUT	5401	Request time limit exceeded
ERR_CALENDAR_NO_DATA	5402	Country is not found
Working with databases		
ERR_DATABASE_ERROR	5601	Generic error
ERR_DATABASE_LOGIC	5602	SQLite internal logic error
ERR_DATABASE_PERM	5603	Access denied
ERR_DATABASE_ABORT	5604	Callback routine requested abort
ERR_DATABASE_BUSY	5605	Database file locked
ERR_DATABASE_LOCKED	5606	Database table locked
ERR_DATABASE_NOMEM	5607	Insufficient memory for completing operation

Constant	Code	Description
ERR_DATABASE_READONLY	5608	Attempt to write to readonly database
ERR_DATABASE_INTERRUPT	5609	Operation terminated by sqlite3_interrupt()
ERR_DATABASE_IOERR	5610	Disk I/O error
ERR_DATABASE_CORRUPT	5611	Database disk image corrupted
ERR_DATABASE_NOTFOUND	5612	Unknown operation code in sqlite3_file_control()
ERR_DATABASE_FULL	5613	Insertion failed because database is full
ERR_DATABASE_CANTOPEN	5614	Unable to open the database file
ERR_DATABASE_PROTOCOL	5615	Database lock protocol error
ERR_DATABASE_EMPTY	5616	Internal use only
ERR_DATABASE_SCHEMA	5617	Database schema changed
ERR_DATABASE_TOOBIG	5618	String or BLOB exceeds size limit
ERR_DATABASE_CONSTRAINT	5619	Abort due to constraint violation
ERR_DATABASE_MISMATCH	5620	Data type mismatch
ERR_DATABASE_MISUSE	5621	Library used incorrectly
ERR_DATABASE_NOLFS	5622	Uses OS features not supported on host
ERR_DATABASE_AUTH	5623	Authorization denied
ERR_DATABASE_FORMAT	5624	Not used
ERR_DATABASE_RANGE	5625	Bind parameter error, incorrect index
ERR_DATABASE_NOTADB	5626	File opened that is not database file
Matrix and Vector Methods		
ERR_MATRIX_INTERNAL	5700	Internal error of the matrix/vector executing subsystem
ERR_MATRIX_NOT_INITIALIZED	5701	Matrix/vector not initialized
ERR_MATRIX_INCONSISTENT	5702	Inconsistent size of matrices/vectors in operation
ERR_MATRIX_INVALID_SIZE	5703	Invalid matrix/vector size

Constant	Code	Description
ERR_MATRIX_INVALID_TYPE	5704	Invalid matrix/vector type
ERR_MATRIX_FUNC_NOT_ALLOWED	5705	Function not available for this matrix/vector
ERR_MATRIX_CONTAINS_NAN	5706	Matrix/vector contains non-numbers (Nan/Inf)
ONNX models		
ERR_ONNX_INTERNAL	5800	ONNX internal error
ERR_ONNX_NOT_INITIALIZED	5801	ONNX Runtime API initialization error
ERR_ONNX_NOT_SUPPORTED	5802	Property or value not supported by MQL5
ERR_ONNX_RUN_FAILED	5803	ONNX runtime API run error
ERR_ONNX_INVALID_PARAMETERS_COUNT	5804	Invalid number of parameters passed to OnnxRun
ERR_ONNX_INVALID_PARAMETER	5805	Invalid parameter value
ERR_ONNX_INVALID_PARAMETER_TYPE	5806	Invalid parameter type
ERR_ONNX_INVALID_PARAMETER_SIZE	5807	Invalid parameter size
ERR_ONNX_WRONG_DIMENSION	5808	Tensor dimension not set or invalid
User errors		
ERR_USER_ERROR_FIRST	65536	User defined errors start with this code

See also

Trade Server Return Codes

Input and Output Constants

Constants:

- File opening flags
- File properties
- Positioning inside a file
- Code page usage
- MessageBox

File Opening Flags

File opening flag values specify the file access mode. Flags are defined as follows:

Identifier	Value	Description
FILE_READ	1	File is opened for reading. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_WRITE	2	File is opened for writing. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_BIN	4	Binary read/write mode (without string to string conversion). Flag is used in FileOpen().
FILE_CSV	8	CSV file (all its elements are converted to strings of the appropriate type, Unicode or ANSI, and separated by separator). Flag is used in FileOpen().
FILE_TXT	16	Simple text file (the same as csv file, but without taking into account the separators). Flag is used in FileOpen().
FILE_ANSI	32	Strings of ANSI type (one byte symbols). Flag is used in FileOpen().
FILE_UNICODE	64	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen().
FILE_SHARE_READ	128	Shared access for reading from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_SHARE_WRITE	256	Shared access for writing from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_REWRITE	512	Possibility for the file rewrite using functions FileCopy() and FileMove(). The file should exist or should be opened for writing, otherwise the file will not be opened.
FILE_COMMON	4096	The file path in the common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen(), FileCopy(), FileMove() and in FileIsExist() functions.

One or several flags can be specified when opening a file. This is a combination of flags. The combination of flags is written using the sign of logical OR (|), which is positioned between enumerated flags. For example, to open a file in CSV format for reading and writing at the same time, specify the combination FILE_READ|FILE_WRITE|FILE_CSV.

Example:

```
   int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

There are some specific features of work when you specify read and write flags:

- If FILE_READ is specified, an attempt is made to open an existing file. If a file does not exist, file opening fails, a new file is not created.
- FILE_READ|FILE_WRITE – a new file is created if the file with the specified name does not exist.
- FILE_WRITE – the file is created again with a zero size.

When opening a file, specification of FILE_WRITE and/or FILE_READ is required.

Flags that define the type of reading of an open file possess priority. The highest flag is FILE_CSV, then goes FILE_BIN, and FILE_TXT is of lowest priority. Thus, if several flags are specified at the same time, (FILE_TXT|FILE_CSV or FILE_TXT|FILE_BIN or FILE_BIN|FILE_CSV), the flag with the highest priority will be used.

Flags that define the type of encoding also have priority. FILE_UNICODE is of a higher priority than FILE_ANSI. So if you specify combination FILE_UNICODE|FILE_ANSI, flag FILE_UNICODE will be used.

If neither FILE_UNICODE nor FILE_ANSI is indicated, FILE_UNICODE is implied. If neither FILE_CSV, nor FILE_BIN, nor FILE_TXT is specified, FILE_CSV is implied.

If a file is opened for reading as a text file (FILE_TXT or FILE_CSV), and at the file beginning a special two-byte indication `0xff,0xfe` is found, the encoding flag will be FILE_UNICODE, even if FILE_ANSI is specified.

See also

File Functions

File Properties

The FileGetInteger() function is used for obtaining file properties. The identifier of the required property from the ENUM_FILE_PROPERTY_INTEGER enumeration is passed to it during call.

ENUM_FILE_PROPERTY_INTEGER

ID	ID description
FILE_EXISTS	Check the existence
FILE_CREATE_DATE	Date of creation
FILE_MODIFY_DATE	Date of the last modification
FILE_ACCESS_DATE	Date of the last access to the file
FILE_SIZE	File size in bytes
FILE_POSITION	Position of a pointer in the file
FILE_END	Get the end of file sign
FILE_LINE_END	Get the end of line sign
FILE_IS_COMMON	The file is opened in a shared folder of all terminals (see FILE_COMMON)
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)
FILE_IS_READABLE	The opened file is readable (see FILE_READ)
FILE_IS_WRITABLE	The opened file is writable (see FILE_WRITE)

The FileGetInteger() function has two different options of call. In the first option, for getting properties of a file, its handle is specified, which is obtained while opening the file using the FileOpen() function. This option allows getting all properties of a file.

The second option of the FileGetInteger() function returns values of file properties by the file name. Using this option, only the following general properties can be obtained:

- FILE_EXISTS – existence of a file with a specified name
- FILE_CREATE_DATE – date of creation of the file with the specified name
- FILE_MODIFY_DATE – date of modification of the file with the specified name
- FILE_ACCESS_DATE – date of the last access to the file with the specified name
- FILE_SIZE – size of the file with the specified name

When trying to get properties other than specified above, the second option of FileGetInteger() call will return an error.

Positioning Inside a File

Most of file functions are associated with data read/write operations. At the same time, using the FileSeek() you can specify the position of a file pointer to a position inside the file, from which the next read or write operation will be performed. The ENUM_FILE_POSITION enumeration contains valid pointer positions, relative to which you can specify the shift in bytes for the next operation.

ENUM_FILE_POSITION

Identifier	Description
SEEK_SET	File beginning
SEEK_CUR	Current position of a file pointer
SEEK_END	File end

See also

FileIsEnding, FileIsLineEnding

Using a Codepage in String Conversion Operations

When converting string variables into arrays of char type and back, the encoding that by default corresponds to the current ANSI of Windows operating system (CP_ACP) is used in MQL5. If you want to specify a different type of encoding, it can be set as additional parameter for the CharArrayToString(), StringToCharArray() and FileOpen() functions.

The table lists the built-in constants for some of the most popular code pages. Not mentioned code pages can be specified by a code corresponding to the page.

Built-in Constants of Codepages

Constant	Value	Description
CP_ACP	0	The current Windows ANSI code page.
CP_OEMCP	1	The current system OEM code page.
CP_MACCP	2	The current system Macintosh code page. **Note:** This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unicode for encoding.
CP_THREAD_ACP	3	The Windows ANSI code page for the current thread.
CP_SYMBOL	42	Symbol code page
CP_UTF7	65000	UTF-7 code page.
CP_UTF8	65001	UTF-8 code page.

See also

Client Terminal Properties

Constants of the MessageBox Dialog Window

This section contains return codes of the MessageBox() function. If a message window has a Cancel button, the function returns IDCANCEL, in case if the ESC key or the Cancel button is pressed. If there is no Cancel button in the message window, the pressing of ESC does not give any effect.

Constant	Value	Description
IDOK	1	"OK" button has been pressed
IDCANCEL	2	"Cancel" button has been pressed
IDABORT	3	"Abort" button has been pressed
IDRETRY	4	"Retry" button has been pressed
IDIGNORE	5	"Ignore" button has been pressed
IDYES	6	"Yes" button has been pressed
IDNO	7	"No" button has been pressed
IDTRYAGAIN	10	"Try Again" button has been pressed
IDCONTINUE	11	"Continue" button has been pressed

The main flags of the MessageBox() function define contents and behavior of the dialog window. This value can be a combination of the following flag groups:

Constant	Value	Description
MB_OK	0x00000000	Message window contains only one button: OK. Default
MB_OKCANCEL	0x00000001	Message window contains two buttons: OK and Cancel
MB_ABORTRETRYIGNORE	0x00000002	Message window contains three buttons: Abort, Retry and Ignore
MB_YESNOCANCEL	0x00000003	Message window contains three buttons: Yes, No and Cancel
MB_YESNO	0x00000004	Message window contains two buttons: Yes and No
MB_RETRYCANCEL	0x00000005	Message window contains two buttons: Retry and Cancel
MB_CANCELTRYCONTINUE	0x00000006	Message window contains three buttons: Cancel, Try Again, Continue

To display an icon in the message window it is necessary to specify additional flags:

Constant	Value	Description
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	The STOP sign icon
MB_ICONQUESTION	0x00000020	The question sign icon
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	The exclamation/warning sign icon
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	The encircled **i** sign

Default buttons are defined by the following flags:

Constant	Value	Description
MB_DEFBUTTON1	0x00000000	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified
MB_DEFBUTTON2	0x00000100	The second button is default
MB_DEFBUTTON3	0x00000200	The third button is default
MB_DEFBUTTON4	0x00000300	The fourth button is default

MQL5 Programs

For the mql5-program to operate, it must be compiled (Compile button or F7 key). Compilation should pass without errors (some warnings are possible; they should be analyzed). At this process, an executable file with the same name and with EX5 extension must be created in the corresponding directory, terminal_dir\MQL5\Experts, terminal_dir\MQL5\indicators or terminal_dir\MQL5\scripts. This file can be run.

Operating features of MQL5 programs are described in the following sections:

- Program running – order of calling predefined event-handlers.
- Testing trading strategies – operating features of MQL5 programs in the Strategy Tester.
- Client terminal events – description of events, which can be processed in programs.
- Call of imported functions – description order, allowed parameters, search details and call agreement for imported functions.
- Runtime errors – getting information about runtime and critical errors.

Expert Advisors, custom indicators and scripts are attached to one of opened charts by Drag'n'Drop method from the Navigator window.

For an expert Advisor to stop operating, it should be removed from a chart. To do it select "Expert list" in chart context menu, then select an Expert Advisor from list and click "Remove" button. Operation of Expert Advisors is also affected by the state of the "AutoTrading" button.

In order to stop a custom indicator, it should be removed from a chart.

Custom indicators and Expert Advisors work until they are explicitly removed from a chart; information about attached Expert Advisors and Indicators is saved between client terminal sessions.

Scripts are executed once and are deleted automatically upon operation completion or change of the current chart state, or upon client terminal shutdown. After the restart of the client terminal scripts are not started, because the information about them is not saved.

Maximum one Expert Advisor, one script and unlimited number of indicators can operate in one chart.

Services do not require to be bound to a chart to work and are designed to perform auxiliary functions. For example, in a service, you can create a custom symbol, open its chart, receive data for it in an endless loop using the network functions and constantly update it.

Program Running

Each script, each service and each Expert Advisor runs in its own separate thread. All indicators calculated on one symbol, even if they are attached to different charts, work in the same thread. Thus, all indicators on one symbol share the resources of one thread.

All other actions associated with a symbol, like processing of ticks and history synchronization, are also consistently performed in the same thread with indicators. This means that if an infinite action is performed in an indicator, all other events associated with its symbol will never be performed.

When running an Expert Advisor, make sure that it has an actual trading environment and can access the history of the required symbol and period, and synchronize data between the terminal and the server. For all these procedures, the terminal provides a start delay of no more than 5 seconds, after which the Expert Advisor will be started with available data. Therefore, in case there is no connection to the server, this may lead to a delay in the start of an Expert Advisor.

The below table contains a brief summary of MQL5 programs:

Program	Running	Note
Service	A separate thread, the number of threads for services is equal to the number of services	A looped service cannot break running of other programs
Script	A separate thread, the number of threads for scripts is equal to the number of scripts	A looped script cannot break running of other programs
Expert Advisor	A separate thread, the number of threads for Expert Advisors is equal to the number of Expert Advisors	A looped Expert Advisor cannot break running of other programs
Indicator	One thread for all indicators on a symbol. The number of threads is equal to the number of symbols with indicators	An infinite loop in one indicator will stop all other indicators on this symbol

Right after a program is attached to a chart, it is uploaded to the client terminal memory, as well as global variable are initialized. If some global variable of the class type has a constructor, this constructor will be called during initialization of global variables.

After that the program is waiting for an event from the client terminal. Each mql5-program should have at least one event-handler, otherwise the loaded program will not be executed. Event handlers have predefined names, parameters and return types.

Type	Function name	Parameters	Application	Comment
int	OnInit	none	Expert Advisors and indicators	Init event handler. It allows to use the void return type.

Type	Function name	Parameters	Application	Comment
void	OnDeinit	const int reason	Expert Advisors and indicators	Deinit event handler.
void	OnStart	none	scripts and services	Start event handler.
int	OnCalculate	const int rates_total, const int prev_calculated, const datetime &Time[], const double &Open[], const double &High[], const double &Low[], const double &Close[], const long &TickVolume[], const long &Volume[], const int &Spread[]	indicators	Calculate event handler for all prices.
int	OnCalculate	const int rates_total, const int prev_calculated, const int begin, const double &price[]	indicators	Calculate event handler on the single data array. Indicator cannot have two event handlers simultaneously. In this case the only one event handler will work on the data array.
void	OnTick	none	Expert Advisors	NewTick event handler. While the event of a new tick receipt is being processed, no other events of this type are received.
void	OnTimer	none	Expert Advisors and indicators	Timer event handler.
void	OnTrade	none	Expert Advisors	Trade event handler.
double	OnTester	none	Expert Advisors	Tester event handler.
void	OnChartEvent	const int id, const long &lparam, const double &dparam, const string &sparam	Expert Advisors and indicators	ChartEvent event handler.
void	OnBookEvent	const string &symbol_name	Expert Advisors and indicators	BookEvent event handler.

A client terminal sends new events to the corresponding open charts. Events can also be generated by charts (chart events) or mql5-programs (custom events). Generation of events of creation or deletion of graphical objects on a chart can be enabled or disabled by setting CHART_EVENT_OBJECT_CREATE and CHART_EVENT_OBJECT_DELETE chart properties. Each MQL5 program and each chart has its own queue of events, where all new incoming events are added.

A program receives only events from the chart it runs on. All events are processed one after another in the order they are received. If a queue already has a NewTick event, or this event is currently being processed, then the new NewTick event is not placed in the queue of the MQL5 program. Similarly, if ChartEvent is already enqueued, or this event is being processed, no new event of this kind is enqueued. The timer events are handled the same way – if the Timer event is in the queue or being handled, the new timer event is not enqueued.

Event queues have a limited but sufficient size, so that the queue overflow for well written programs is unlikely. In case of queue overflow, new events are discarded without queuing.

It is strongly recommended not to use infinite loops to handle events. Possible exceptions are scripts and services handling a single Start event.

Libraries do not handle any events.

Functions prohibited in Indicators and Expert Advisors

Indicators, scripts and Expert Advisors are executable programs written in MQL5. They are designed for different types of tasks. Therefore there are some restrictions on the use of certain functions, depending on the type of program. The following functions are prohibited in indicators:

- OrderCalcMargin();
- OrderCalcProfit();
- OrderCheck();
- OrderSend();
- SendFTP();
- Sleep();
- ExpertRemove();
- MessageBox().

All functions designed for indicators are prohibited in Expert Advisors and scripts:

- SetIndexBuffer();
- IndicatorSetDouble();
- IndicatorSetInteger();
- IndicatorSetString();
- PlotIndexSetDouble();
- PlotIndexSetInteger();
- PlotIndexSetString();

- PlotIndexGetInteger.

The library is not an independent program and is executed in the context of the MQL5 program that has called it: script, indicator or Expert Advisor. Accordingly, the above restrictions apply to the called library.

Functions prohibited in services

Services do not accept any events, as they are not bound to a chart. The following functions are prohibited in services:

ExpertRemove();

EventSetMillisecondTimer();

EventSetTimer();

EventKillTimer();

SetIndexBuffer();

IndicatorSetDouble();

IndicatorSetInteger();

IndicatorSetString();

PlotIndexSetDouble();

PlotIndexSetInteger();

PlotIndexSetString();

PlotIndexGetInteger();

Loading and Unloading of Indicators

Indicators are loaded in the following cases:

- an indicator is attached to a chart;
- terminal start (if the indicator was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the indicator attached to a chart is specified in the template);
- change of a profile (if the indicator is attached to one of the profile charts);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- change of the account to which the terminal is connected;
- after the successful recompilation of an indicator (if the indicator was attached to a chart);
- change of input parameters of the indicator.

Indicators are unloaded in the following cases:

- when detaching an indicator from a chart;
- terminal shutdown (if the indicator was attached to a chart);
- loading of a template (if an indicator is attached to a chart);

- closing of a chart, to which the indicator was attached;
- change of a profile (if the indicator is attached to one of charts of the changed profile);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- change of the account to which the terminal is connected;
- change of input parameters of the indicator.

Loading and Unloading of Expert Advisors

Expert Advisors are loaded in the following cases:

- when attaching an Expert Advisor to a chart;
- terminal start (if the Expert Advisor was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the Expert Advisor attached to the chart is specified in the template);
- change of a profile (if the Expert Advisor is attached to the one of the profile charts);
- connection to an account, even if the account number is the same (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server).

Expert Advisors are unloaded in the following cases:

- when detaching an Expert Advisor from a chart;
- if a new Expert Advisor is attached to a chart, if another Expert Advisor has been attached already, this Expert Advisor is unloaded.
- terminal shutdown (if the Expert Advisor was attached to a chart);
- loading of a template (if an Expert Advisor is attached to the chart);
- close of a chart, to which the Expert Advisor is attached.
- change of a profile (if the Expert Advisor is attached to one of charts of the changed profile);
- change of the account to which the terminal is connected (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server);
- calling the ExpertRemove() function.

In case the symbol or timeframe of a chart, to which the Expert Advisor is attached, changes, Expert Advisors are not loaded or unloaded. In this case client terminal subsequently calls OnDeinit() handlers on the old symbol/timeframe and OnInit() on the new symbol/timeframe (if they are such), values of global variables and static variables are not reset. All events, which have been received for the Expert Advisor before the initialization is completed (OnInit() function) are skipped.

Loading and Unloading of Scripts

Scripts are loaded immediately after they are attached to a chart and unloaded immediately after they complete their operation. OnInit() and OnDeinit() are not called for scripts.

When a program is unloaded (deleted from a chart) the client terminal performs deinitialization of global variables and deletes the events queue. In this case deinitialization means reset of all the string-type variables, deallocation of dynamical array objects and call of their destructors if they are available.

Loading and Unloading services

Services are loaded right after starting the terminal if they were launched at the moment of the terminal shutdown. Services are unloaded immediately after completing their work.

Services have a single OnStart() handler, in which you can implement an endless data receiving and handling loop, for example creating and updating custom symbols using the network functions.

Unlike Expert Advisors, indicators and scripts, services are not bound to a specific chart, therefore a separate mechanism is provided to launch them. A new service instance is created in the Navigator using the "Add Service" command. A service instance can be launched, stopped and removed using the appropriate instance menu. To manage all instances, use the service menu.

For a better understanding of the Expert Advisor operation we recommend to compile the code of the following Expert Advisor and perform actions of load/unload, template change, symbol change, timeframe change etc:

Example:

```
//+------------------------------------------------------------------+
//|                                                   TestExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
  {
public:
   CTestClass() { Print("CTestClass constructor"); }
   ~CTestClass() { Print("CTestClass destructor"); }
  };
CTestClass global;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   Print("Initialization");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print("Deinitialization with reason",reason);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
```

See also

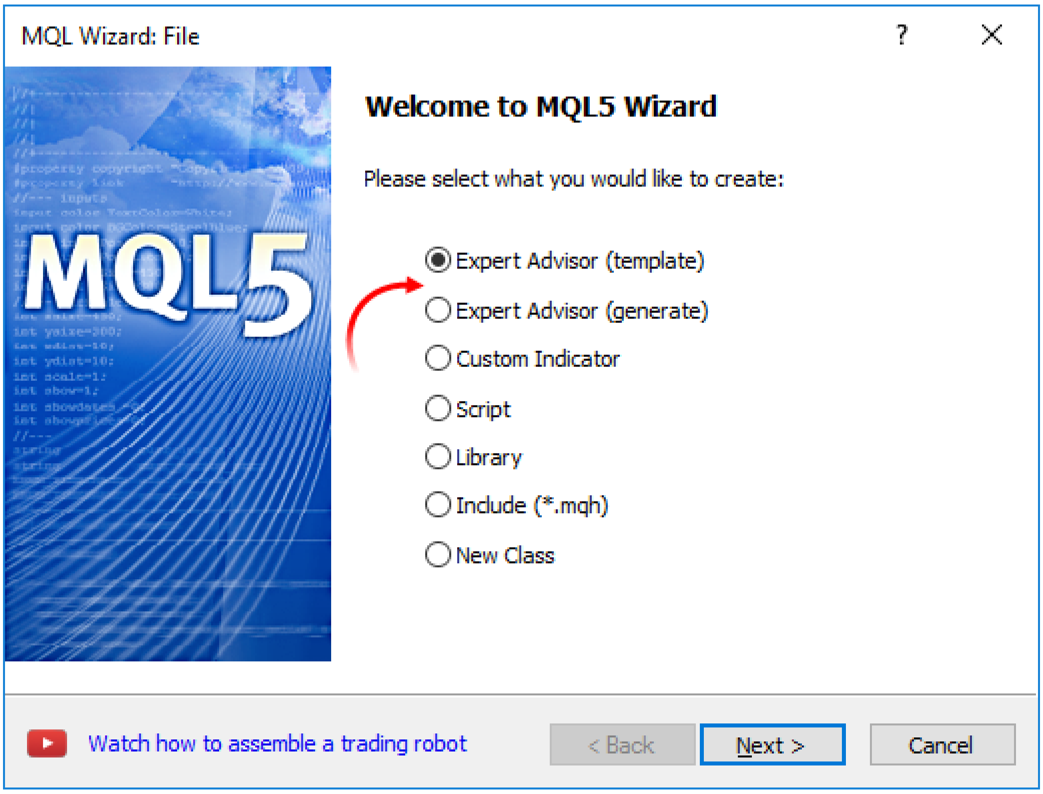
Client terminal events, Event handlers

Trade Permission

Trade Automation

MQL5 language provides a special group of trade functions designed for developing automated trading systems. Programs developed for automated trading with no human intervention are called Expert Advisors or trading robots. In order to create an Expert Advisor in MetaEditor, launch MQL5 Wizard and select one of the two options:

- Expert Advisor (template) – allows you to create a template with ready-made event handling functions that should be supplemented with all necessary functionality by means of programming.
- Expert Advisor (generate) – allows you to develop a full-fledged trading robot simply by selecting the necessary modules: trading signals module, money management module and trailing stop module.


MQL Wizard: File

Welcome to MQL5 Wizard

Please select what you would like to create:

● Expert Advisor (template)
○ Expert Advisor (generate)
○ Custom Indicator
○ Script
○ Library
○ Include (*.mqh)
○ New Class

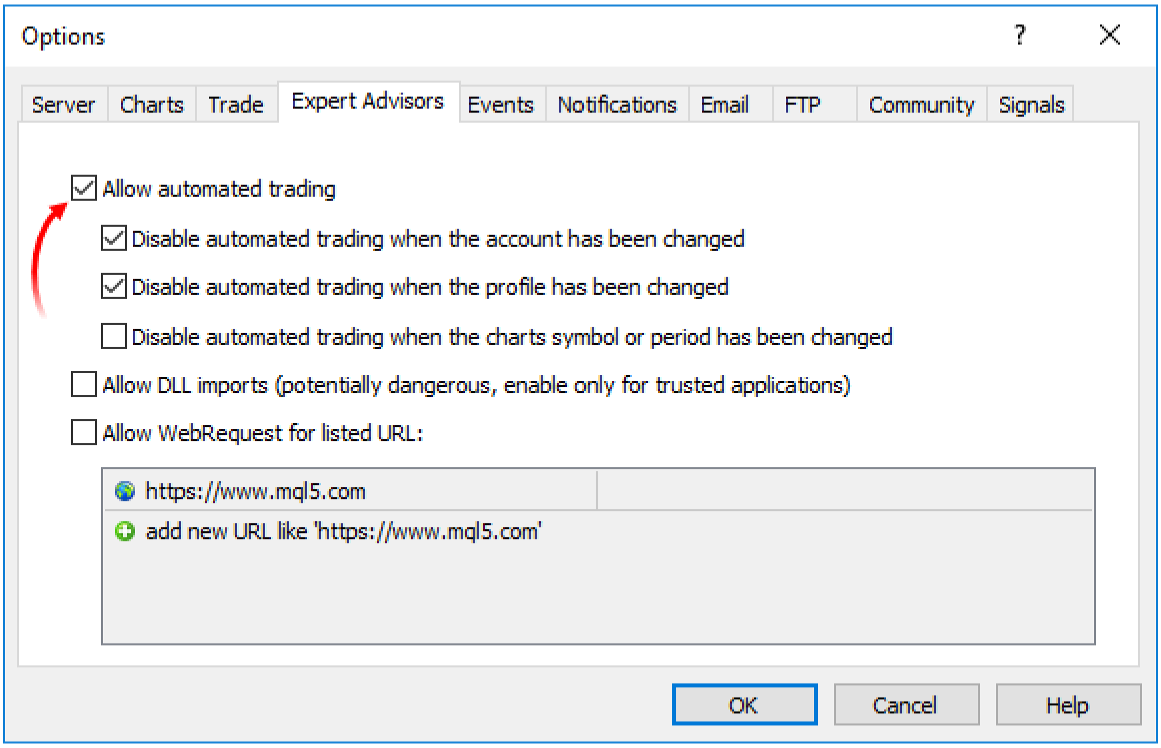
Trading functions can work only in Expert Advisors and scripts. Trading is not allowed for indicators.

Checking for Permission to Perform Automated Trading

In order to develop a reliable Expert Advisor capable of working without human intervention, it is necessary to arrange a set of important checks. First, we should programmatically check if trading is allowed at all. This is a basic check that is indispensable when developing any automated system.

Checking for permission to perform automated trading in the terminal

The terminal settings provide you with an ability to allow or forbid automated trading for all programs.



☑ Allow automated trading

☑ Disable automated trading when the account has been changed

☑ Disable automated trading when the profile has been changed

☐ Disable automated trading when the charts symbol or period has been changed

☐ Allow DLL imports (potentially dangerous, enable only for trusted applications)

☐ Allow WebRequest for listed URL:

https://www.mql5.com

add new URL like 'https://www.mql5.com'

You can switch automated trading option right on the terminal's Standard panel:

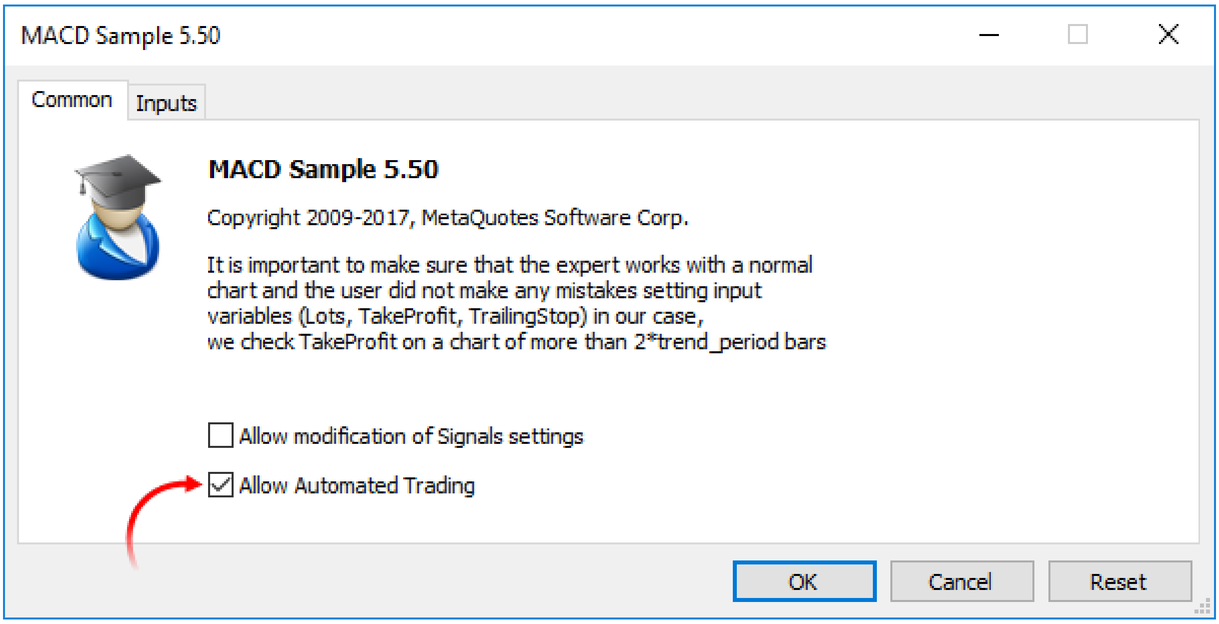
- AutoTrading - automated trading enabled, trading functions in launched applications are allowed for use.
- AutoTrading - automated trading disabled, running applications are unable to execute trading functions.

Sample check:

```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
   Alert("Check if automated trading is allowed in the terminal settings!");
```

Checking if trading is allowed for a certain running Expert Advisor/script

You can allow or forbid automated trading for a certain program when launching it. To do this, use the special check box in the program properties.



MACD Sample 5.50

Copyright 2009-2017, MetaQuotes Software Corp.

It is important to make sure that the expert works with a normal
chart and the user did not make any mistakes setting input
variables (Lots, TakeProfit, TrailingStop) in our case,
we check TakeProfit on a chart of more than 2*trend_period bars

☐ Allow modification of Signals settings

☑ Allow Automated Trading

Sample check:

```
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
      Alert("Check if automated trading is allowed in the terminal settings!");
   else
     {
      if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
         Alert("Automated trading is forbidden in the program settings for ",__FILE__)
     }
```

Checking if trading is allowed for any Expert Advisors/scripts for the current account

Automated trading can be disabled at the trade server side. Sample check:

```
   if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT))
      Alert("Automated trading is forbidden for the account ",AccountInfoInteger(ACCOU
      " at the trade server side");
```

If automated trading is disabled for a trading account, trading operations of Expert Advisors/scripts are not executed.

Checking if trading is allowed for the current account

In some cases, any trading operations are disabled for a certain trading account – neither manual nor automated trading can be performed. Sample check when an investor password has been used to connect to a trading account:

```
   if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
      Comment("Trading is forbidden for the account ",AccountInfoInteger(ACCOUNT_LOGI
            ".\n Perhaps an investor password has been used to connect to the trading
            "\n Check the terminal journal for the following entry:",
                    "\n\'",AccountInfoInteger(ACCOUNT_LOGIN),"\': trading has been disabled -
```

AccountInfoInteger(ACCOUNT_TRADE_ALLOWED) may return **false** in the following cases:

- no connection to the trade server. That can be checked using TerminalInfoInteger(TERMINAL_CONNECTED);
- trading account switched to read-only mode (sent to the archive);
- trading on the account is disabled at the trade server side;
- connection to a trading account has been performed in Investor mode.

See also

Client Terminal Properties, Account Properties, Properties of a Running MQL5 Program

Client Terminal Events

Init

Immediately after the client terminal loads a program (an Expert Advisor or custom indicator) and starts the process of initialization of global variables, the Init event will be sent, which will be processed by OnInit() event handler, if there is such. This event is also generated after a financial instrument and/or chart timeframe is changed, after a program is recompiled in MetaEditor, after input parameters are changed from the setup window of an Expert Advisor or a custom indicator. An Expert Advisor is also initialized after the account is changed. The Init event is not generated for scripts.

Deinit

Before global variables are deinitialized and the program (Expert Advisor or custom indicator) is unloaded, the client terminal sends the Deinit event to the program. Deinit is also generated when the client terminal is closed, when a chart is closed, right before the security and/or timeframe is changed, at a successful program re-compilation, when input parameters are changed, and when account is changed.

The deinitialization reason can be obtained from the parameter, passed to the OnDeinit() function. The OnDeinit() function run is restricted to 2.5 seconds. If during this time the function hasn't been completed, then it is forcibly terminated. The Deinit event is not generated for scripts.

Start

Start is a special event for launching a script or a service after loading it. It is handled by the OnStart function. The Start event is not passed to EAs and custom indicators.

NewTick

The NewTick event is generated if there are new quotes, it is processed by OnTick() of Expert Advisors attached. In case when OnTick function for the previous quote is being processed when a new quote is received, the new quote will be ignored by an Expert Advisor, because the corresponding event will not enqueued.

All new quotes that are received while the program is running are ignored until the OnTick() is completed. After that the function will run only after a new quote is received. The NewTick event is generated irrespective of whether automated trade is allowed or not ("Allow/prohibit Auto trading" button). The prohibition of automated trading denotes only that sending of trade requests from an Expert Advisor is not allowed, while the Expert Advisor keeps working.

The prohibition of automated trading by pressing the appropriate button will not stop the current execution of the OnTick() function.

Calculate

The Calculate event is generated only for indicators right after the Init event is sent and at any change of price data. It is processed by the OnCalculate function.

Timer

The Timer event is periodically generated by the client terminal for the Expert Advisor that has activated the timer by the EventSetTimer function. Usually, this function is called by OnInit. Timer event processing is performed by the OnTimer function. After the operation of the Expert Advisor is completed, it is necessary to destroy the timer using the EventKillTimer function, which is usually called in the OnDeinit function.

Trade

The Trade event is generated when a trade operation is completed on a trade server. The Trade event is handled by the OnTrade() function for the following trade operations:

- sending, modifying or removing of a pending order;
- cancellation of a pending order with not enough of money or expiration;
- activation of a pending order;
- opening, adding or closing a position (or part of the position);
- modifying of the open position (change stops – Stop Loss and/or Take Profit).

TradeTransaction

When performing some definite actions on a trade account, its state changes. Such actions include:

- Sending a trade request from any MQL5 application in the client terminal using OrderSend and OrderSendAsync functions and its further execution;
- Sending a trade request via the terminal graphical interface and its further execution;
- Pending orders and stop orders activation on the server;
- Performing operations on a trade server side.

The following trade transactions are performed as a result of these actions:

- handling a trade request;
- changing open orders;
- changing orders history;
- changing deals history;
- changing positions.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. All these actions are trade transactions. Arrival of such a transaction at the terminal is a TradeTransaction event. This event is handled by OnTradeTransaction function.

Tester

The Tester event is generated after testing of an Expert Advisor on history data is over. The event is handled by the OnTester() function.

TesterInit

The TesterInit event is generated with the start of optimization in the strategy tester before the first optimization pass. The TesterInit event is handled by the OnTesterInit() function.

TesterPass

The TesterPass event is generated when a new data frame is received. The TesterPass event is handled by the OnTesterPass() function.

TesterDeinit

The TesterDeinit event is generated after the end of optimization of an Expert Advisor in the strategy tester. The TesterDeinit event is handled by the OnTesterDeinit() function.

ChartEvent

The ChartEvent event is generated by the client terminal when a user is working with a chart:

- keystroke, when the chart window is in focus;
- graphical object created
- graphical object deleted
- mouse press on the graphical object of the chart
- move of the graphical object using the mouse
- end of text editing in LabelEdit.

Also there is a custom event ChartEvent, which can be sent to an Expert Advisor by any mql5 program by using the EventChartCustom function. The event is processed by the OnChartEvent function.

BookEvent

The BookEvent event is generated by the client terminal after the Depth Of Market is changed; it is processed by the OnBookEvent function. To start generation of BookEvent for the specified symbol, it is necessary to subscribe the symbol to this event by using the MarketBookAdd function.

To unsubscribe from BookEvent for a specified symbol, it is necessary to call the MarketBookRelease function. The BookEvent event is a broadcasting-type event - it means that it is sufficient to subscribe just one Expert Advisor for this event, and all other Expert Advisors that have the OnBookEvent event handler, will receive it. That's why it is necessary to analyze the symbol name, which is passed to a handler as a parameter.

See also

Event handlers, Program running

Resources

Using graphics and sound in MQL5 programs

Programs in MQL5 allow working with sound and graphic files:

- PlaySound() plays a sound file;
- ObjectCreate() allows creating user interfaces using graphical objects OBJ_BITMAP and OBJ_BITMAP_LABEL.

PlaySound()

Example of call of the PlaySound() function:

```
//+------------------------------------------------------------------+
//| Calls standard OrderSend() and plays a sound                     |
//+------------------------------------------------------------------+
void OrderSendWithAudio(MqlTradeRequest  &request, MqlTradeResult &result)
  {
  //--- send a request to a server
   OrderSend(request,result);
   //--- if a request is accepted, play sound Ok.wav
   if(result.retcode==TRADE_RETCODE_PLACED) PlaySound("Ok.wav");
   //--- if fails, play alarm from file timeout.wav
   else PlaySound("timeout.wav");
  }
```

The example shows how to play sounds from files 'Ok.wav' and 'timeout.wav', which are included into the standard terminal package. These files are located in the folder **terminal_directory\Sounds**. Here **terminal_directory** is a folder, from which the MetaTrader 5 Client Terminal is started. The location of the terminal directory can be found out from an mql5 program in the following way:

```
//--- Folder, in which terminal data are stored
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

You can use sound files not only from the folder **terminal_directory\Sounds**, but also from any subfolder located in **terminal_data_directory\MQL5.** You can find out the location of the terminal data directory from the terminal menu "File" -> "Open Data Folder" or using program method:

```
//--- Folder, in which terminal data are stored
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

For example, if the Demo.wav sound file is located in terminal_data_directory\MQL5\Files, then call of PlaySound() should be written the following way:

```
//--- play Demo.wav from the folder terminal_directory_data\MQL5\Files\
   PlaySound("\\Files\\Demo.wav");
```

Please note that in the comment the path to the file is written using backslash "\", and in the function "\\" is used.

When specifying the path, always use only the double backslash as a separator, because a single backslash is a control symbol for the compiler when dealing with constant strings and character constants in the program source code.

Call PlaySound() function with NULL parameter to stop playback:

```
//--- call of PlaySound() with NULL parameter stops playback
   PlaySound(NULL);
```

ObjectCreate()

Example of an Expert Advisor, which creates a graphical label (OBJ_BITMAP_LABEL) using the ObjectCreate() function.

```
string label_name="currency_label";        // name of the OBJ_BITMAP_LABEL object
string euro      ="\\Images\\euro.bmp";    // path to the file terminal_data_directory
string dollar    ="\\Images\\dollar.bmp";  // path to the file terminal_data_directory
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create a button OBJ_BITMAP_LABEL, if it hasn't been created yet
   if(ObjectFind(0,label_name)<0)
     {
      //--- trying to create object OBJ_BITMAP_LABEL
      bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
      if(created)
        {
         //--- link the button to the left upper corner of the chart
         ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
         //--- now set up the object properties
         ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
         ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
         //--- reset the code of the last error to 0
         ResetLastError();
         //--- download a picture to indicate the "Pressed" state of the button
         bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
         //--- test the result
         if(!set)
           {
            PrintFormat("Failed to download image from file %s. Error code %d",euro,Ge
           }
         ResetLastError();
         //--- download a picture to indicate the "Unpressed" state of the button
         set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

         if(!set)
```

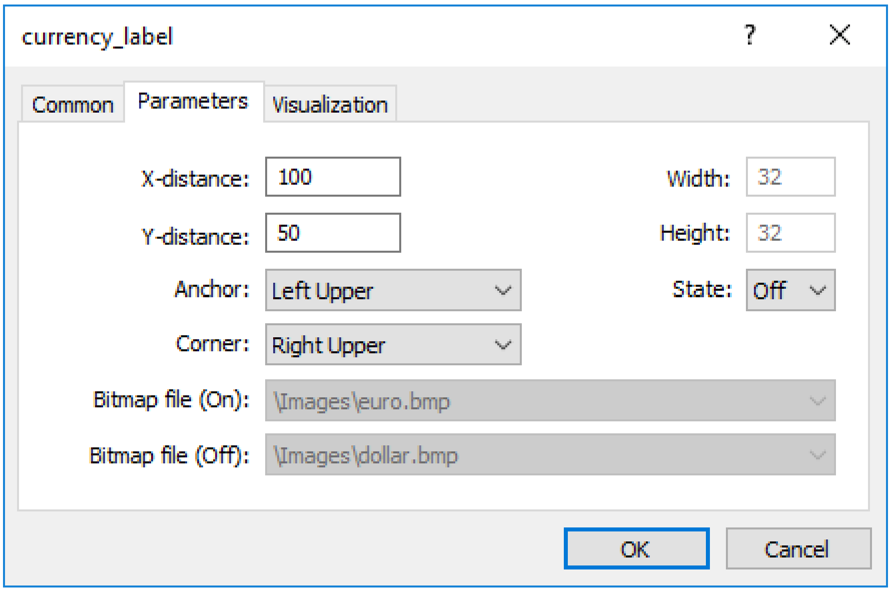
```
            {
             PrintFormat("Failed to download image from file %s. Error code %d",dollar,
            }
          //--- send a command for a chart to refresh so that the button appears immedi
          ChartRedraw(0);
         }
       else
         {
          //--- failed to create an object, notify
          PrintFormat("Failed to create object OBJ_BITMAP_LABEL. Error code %d",GetLast
         }
      }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- delete an object from a chart
   ObjectDelete(0,label_name);
  }
```

Creation and setup of the graphical object named currency_label are carried out in the OnInit() function. The paths to the graphical files are set in global variables **euro** and **dollar**, a double backlash is used for a separator:

```
string euro      ="\\Images\\euro.bmp";    // path to the file terminal_dara_directory
string dollar    ="\\Images\\dollar.bmp";  // path to the file terminal_dara_directory
```

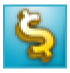
The files are located in the folder **terminal_data_directory**\MQL5\Images.

Object OBJ_BITMAP_LABEL is actually a button, which displays one of the two images, depending on the button state (pressed or unpressed): euro.bmp or dollar.bmp.



currency_label

X-distance: 100
Y-distance: 50
Anchor: Left Upper
Corner: Right Upper
Bitmap file (On): \Images\euro.bmp
Bitmap file (Off): \Images\dollar.bmp
Width: 32
Height: 32
State: Off



The size of the button with a graphical interface is automatically adjusted to the size of the picture. The image is changed by a left mouse button click on the OBJ_BITMAP_LABEL object (**"Disable selection" option must be checked in the properties**). The OBJ_BITMAP object is created the same way - it is used for creating the background with a necessary image.

The value of the OBJPROP_BMPFILE property, which is responsible for the appearance of the objects OBJ_BITMAP and OBJ_BITMAP_LABEL, can be changed dynamically. This allows creating various interactive user interfaces for mql5 programs.

Including resources to executable files during compilation of mql5 programs

An mql5 program may need a lot of different downloadable resources in the form of image and sound files. In order to eliminate the need to transfer all these files when moving an executable file in MQL5, the compiler's directive #resource should be used:

```
#resource path_to_resource_file
```

The #resource command tells the compiler that the resource at the specified path **path_to_resource_file** should be included into the executable EX5 file. Thus all the necessary images and sounds can be located directly in an EX5 file, so that there is no need to transfer separately the files used in it, if you want to run the program on a different terminal. Any EX5 file can contain resources, and any EX5 program can use resources from another EX5 program.

The files in format BMP and WAV are automatically compressed before including them to an EX5 file. This denotes that in addition to the creation of complete programs in MQL5, using resources also allows to reduce the total size of necessary files when using graphics and sounds, as compared to the usual way of MQL5 program writing.

The resource file size must not exceed 128 Mb.

Search for specified resources by a compiler

A resource is inserted using the command #resource "<path to the resource file>"

```
#resource "<path_to_resource_file>"
```

The length of the constant string <path_to_resource_file> must not exceed 63 characters.

The compiler searches for a resource at the specified path in the following order:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, it searches for the resource relative to the directory **terminal_data_directory**\MQL5\,
- if there is no backslash, it searches for the resource relative to the location of the source file, in which the resource is written.

The resource path cannot contain the substrings "..\\" and ":\\".

Examples of resource inclusion:

```
//--- correct specification of resources
#resource "\\Images\\euro.bmp" // euro.bmp is located in terminal_data_directory\MQL5\
#resource "picture.bmp"        // picture.bmp is located in the same directory as the
#resource "Resource\\map.bmp"  // the resource is located in source_file_directory\Res

//--- incorrect specification of resources
#resource ":picture_2.bmp"     // must not contain ":"
#resource "..\\picture_3.bmp"  // must not contain ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.bmp" //more
```

Use of Resources

Resource name

After a resource is declared using the #resource directive, it can be used in any part of a program. The name of the resource is its path without a backslash at the beginning of the line, which sets the path to the resource. To use your own resource in the code, the special sign "::" should be added before the resource name.

Examples:

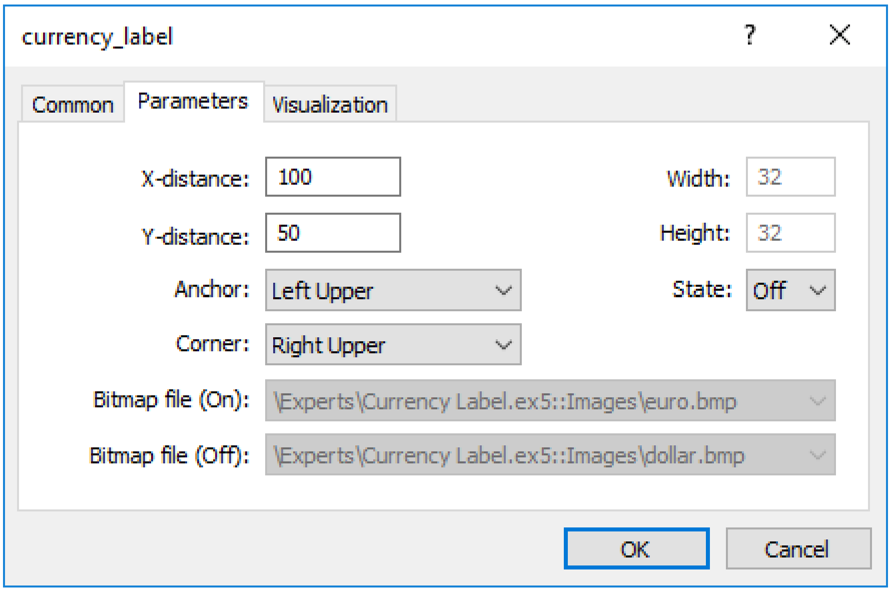
```
//--- examples of resource specification and their names in comments
#resource "\\Images\\euro.bmp"          // resource name - Images\euro.bmp
#resource "picture.bmp"                 // resource name - picture.bmp
#resource "Resource\\map.bmp"           // resource name - Resource\map.bmp
#resource "\\Files\\Pictures\\good.bmp" // resource name - Files\Pictures\good.bmp
#resource "\\Files\\Demo.wav";          // resource name - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";       // resource name - Sounds\thrill.wav"
...

//--- utilization of resources
```

```
ObjectSetString(0,bitmap_name,OBJPROP_BMPFILE,0,"::Images\\euro.bmp");
...
ObjectSetString(0,my_bitmap,OBJPROP_BMPFILE,0,"::picture.bmp");
...
set=ObjectSetString(0,bitmap_label,OBJPROP_BMPFILE,1,"::Files\\Pictures\\good.bmp");
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

It should be noted that when setting images from a resource to the OBJ_BITMAP and OBJ_BITMAP_LABEL objects, the value of the OBJPROP_BMPFILE property cannot be modified manually. For example, for creating OBJ_BITMAP_LABEL we use resources euro.bmp and dollar.bmp.

```
#resource "\\Images\\euro.bmp";    // euro.bmp is located in terminal_data_directory\
#resource "\\Images\\dollar.bmp";  // dollar.bmp is located in terminal_data_directory
```

When viewing the properties of this object, we'll see that the properties BitMap File (On) and BitMap File (Off) are dimmed and cannot be change manually:



X-distance: 100
Y-distance: 50
Anchor: Left Upper
Corner: Right Upper
Bitmap file (On): \Experts\Currency Label.ex5::Images\euro.bmp
Bitmap file (Off): \Experts\Currency Label.ex5::Images\dollar.bmp
Width: 32
Height: 32
State: Off

Using the resources of other mql5 programs

There is another advantage of using resources – in any MQL5 program, resources of another EX5 file can be used. Thus the resources from one EX5 file can be used in many other mql5 programs.

In order to use a resource name from another file, it should be specified as <path_EX5_file_name>::<resource_name>. For example, suppose the Draw_Triangles_Script.mq5 script contains a resource to an image in the file triangle.bmp:

```
 #resource "\\Files\\triangle.bmp"
```

Then its name, for using in the script itself, will look like "Files\triangle.bmp", and in order to use it, "::" should be added to the resource name.

```
//--- using the resource in the script
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp");
```

In order to use the same resource from another program, e.g. from an Expert Advisor, we need to add to the resource name the path to the EX5 file relative to **terminal_data_directory\MQL5\** and the name of the script's EX5 file - **Draw_Triangles_Script.ex5**. Suppose the script is located in the standard folder **terminal_data_directory\MQL5\**Scripts\, then the call should be written the following way:

```
//--- using a resource from a script in an EA
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triangles_Script.e
```

If the path to the executable file is not specified when calling the resource from another EX5, the executable file is searched for in the same folder that contains the program that calls the resource. This means that if an Expert Advisor calls a resource from Draw_Triangles_Script.ex5 without specification of the path, like this:

```
//--- call script resource in an EA without specifying the path
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.ex5::Files\\
```

then the file will be searched for in the folder **terminal_data_directory\MQL5\**Experts\, if the Expert Advisor is located in **terminal_data_directory\MQL5\**Experts\.

Working with custom indicators included as resources

One or several custom indicators may be necessary for the operation of MQL5 applications. All of them can be included into the code of an executable MQL5 program. Inclusion of indicators as resources simplifies the distribution of applications.

Below is an example of including and using SampleIndicator.ex5 custom indicator located in **terminal_data_folder**\MQL5\Indicators\ directory:

```
//+------------------------------------------------------------------+
//|                                                     SampleEA.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#resource "\\Indicators\\SampleIndicator.ex5"
int handle_ind;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   handle_ind=iCustom(_Symbol,_Period,"::Indicators\\SampleIndicator.ex5");
   if(handle_ind==INVALID_HANDLE)
     {
      Print("Expert: iCustom call: Error code=",GetLastError());
      return(INIT_FAILED);
     }
//--- ...
   return(INIT_SUCCEEDED);
  }
```

The case when a custom indicator in OnInit() function creates one or more copies of itself requires special consideration. Please keep in mind that the resource should be specified in the following way: **<path_EX5_file_name>::<resource_name>**.

For example, if SampleIndicator.ex5 indicator is included to SampleEA.ex5 Expert Advisor as a resource, the path to itself specified when calling iCustom() in the custom indicator's initialization function looks the following way: "\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5". When this path is set explicitly, SampleIndicator.ex5 custom indicator is rigidly connected to SampleEA.ex5 Expert Advisor losing ability to work independently.

The path to itself can be received using GetRelativeProgramPath() function. The example of its usage is provided below:

```
//+------------------------------------------------------------------+
//|                                             SampleIndicator.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property indicator_separate_window
#property indicator_plots 0
int handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- the wrong way to provide a link to itself
//--- string path="\\Experts\\SampleEA.ex5::Indicators\\SampleIndicator.ex5";
//--- the right way to receive a link to itself
  string path=GetRelativeProgramPath();
//--- indicator buffers mapping
   handle=iCustom(_Symbol,_Period,path,0,0);
   if(handle==INVALID_HANDLE)
     {
      Print("Indicator: iCustom call: Error code=",GetLastError());
      return(INIT_FAILED);
     }
   else Print("Indicator handle=",handle);
//---
   return(INIT_SUCCEEDED);
  }
///....
//+------------------------------------------------------------------+
//| GetRelativeProgramPath                                           |
//+------------------------------------------------------------------+
string GetRelativeProgramPath()
  {
   int pos2;
//--- get the absolute path to the application
   string path=MQLInfoString(MQL_PROGRAM_PATH);
//--- find the position of "\MQL5\" substring
   int    pos =StringFind(path,"\\MQL5\\");
//--- substring not found - error
   if(pos<0)
      return(NULL);
//--- skip "\MQL5" directory
   pos+=5;
//--- skip extra '\' symbols
   while(StringGetCharacter(path,pos+1)=='\\')
      pos++;
//--- if this is a resource, return the path relative to MQL5 directory
   if(StringFind(path,"::",pos)>=0)
      return(StringSubstr(path,pos));
//--- find a separator for the first MQL5 subdirectory (for example, MQL5\Indicators)
//--- if not found, return the path relative to MQL5 directory
   if((pos2=StringFind(path,"\\",pos+1))<0)
      return(StringSubstr(path,pos));
//--- return the path relative to the subdirectory (for example, MQL5\Indicators)
   return(StringSubstr(path,pos2+1));
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double& price[])
  {
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Resource variables

Resources can be declared using the resource variables and treated as if they are variables of the appropriate type. Declaration format:

```
#resource path_to_the_resource_file as resource_variable_type resource_variable_name
```

Sample declarations:

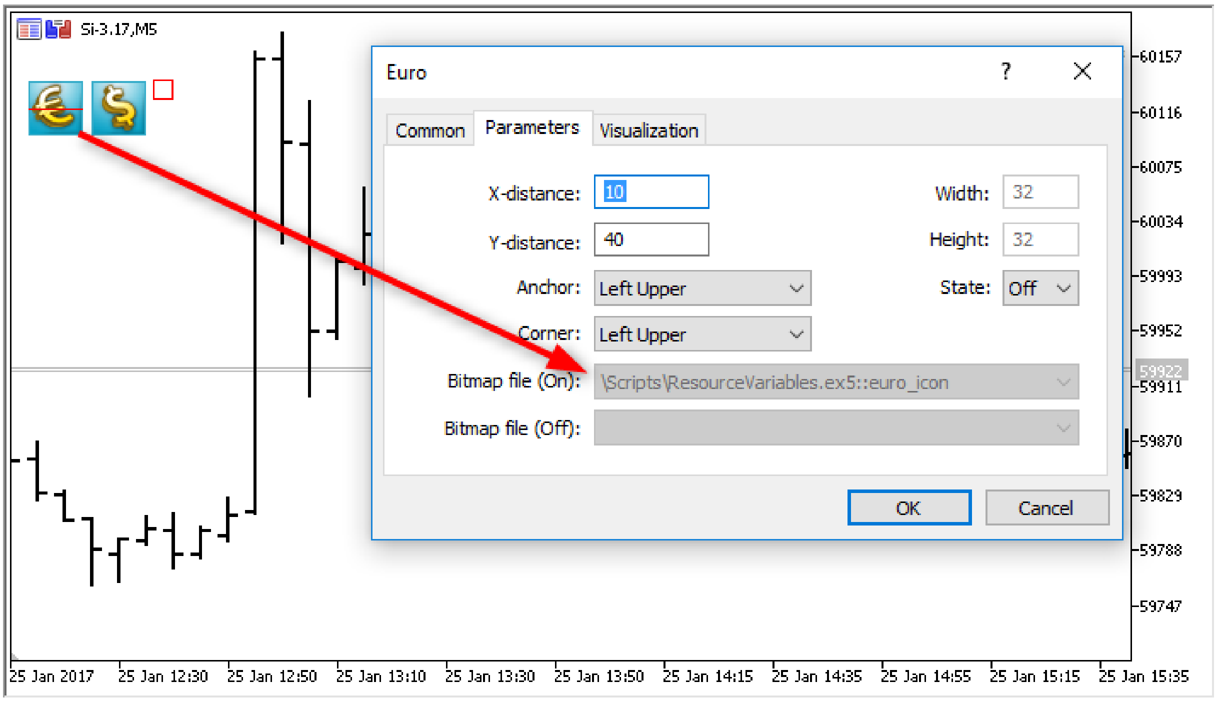
```
#resource "data.bin" as int ExtData[]             // declare the numeric array contai
```

```
#resource "data.bin" as MqlRates ExtData[]          // declare the simple structures arr
//--- strings
#resource "data.txt" as string ExtCode              // declare the string containing the
//--- graphical resources
#resource "image.bmp" as bitmap ExtBitmap[]         // declare the one-dimensional array
#resource "image.bmp" as bitmap ExtBitmap2[][]      // declare the two-dimensional array
```

In case of such declaration, the resource data can be addressed only via the variable, auto addressing via "::<rsource name>" does not work.

```
#resource "\\Images\\euro.bmp" as bitmap euro[][]
#resource "\\Images\\dollar.bmp"
//+------------------------------------------------------------------+
//|  OBJ_BITMAP_LABEL object creation function using the resource    |
//+------------------------------------------------------------------+
void Image(string name,string rc,int x,int y)
  {
   ObjectCreate(0,name,OBJ_BITMAP_LABEL,0,0,0);
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
   ObjectSetString(0,name,OBJPROP_BMPFILE,rc);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- output the size of the image [width, height] stored in euro resource variable
   Print(ArrayRange(euro,1),", ",ArrayRange(euro,0));
//--- change the image in euro - draw the red horizontal stripe in the middle
   for(int x=0;x<ArrayRange(euro,1);x++)
      euro[ArrayRange(euro,1)/2][x]=0xFFFF0000;
//--- create the graphical resource using the resource variable
   ResourceCreate("euro_icon",euro,ArrayRange(euro,1),ArrayRange(euro,0),0,0,ArrayRang
//--- create the Euro graphical label object, to which the image from the euro_icon re
   Image("Euro","::euro_icon",10,40);
//--- another method of applying the resource, we cannot draw do it
   Image("USD","::Images\\dollar.bmp",15+ArrayRange(euro,1),40);
//--- direct method of addressing the euro.bmp resource is unavailable since it has al
   Image("E2","::Images\\euro.bmp",20+ArrayRange(euro,1)*2,40); // execution time erro
  }
```

Script execution result – only two OBJ_BITMAP_LABEL objects out of three ones are created. The image of the first object has the red stripe in the middle.


Si-3.17,M5
Euro
?
Common
Parameters
Visualization
OK
Cancel
60157
60116
60075
60034
59993
59952
59922
59911
59870
59829
59788
59747
25 Jan 2017
25 Jan 12:30
25 Jan 12:50
25 Jan 13:10
25 Jan 13:30
25 Jan 13:50
25 Jan 14:15
25 Jan 14:35
25 Jan 14:55
25 Jan 15:15
25 Jan 15:35

X-distance: 10

Y-distance: 40

Anchor: Left Upper

Corner: Left Upper

Bitmap file (On): \Scripts\ResourceVariables.ex5::euro_icon

Bitmap file (Off):

Width: 32

Height: 32

State: Off

An important advantage of applying the resources is that the resource files are automatically compressed before they are included into an executable EX5 file prior to compilation. Thus, using the resource variables allows you to put all necessary data directly into the executable EX5 file as well as reduce the number and total size of the files compared to the conventional way of writing MQL5 programs.

Using the resource variables is particularly convenient for publishing products in the Market.

Features

- The special *bitmap* resource variable type informs the compiler that the resource is an image. Such variables receive the uint type.
- The *bitmap* type array resource variable may have two dimensions. In this case, the array size is defined as [image_height][image_width]. If an array of one dimension is specified, the number of elements is equal to image_height*image_width.
- When downloading a 24-bit image, the alpha channel component is set to 255 for all the image pixels.
- When downloading a 32-bit image without the alpha channel, the alpha channel component is also set to 255 for all the image pixels.
- When downloading a 32-bit image with the alpha channel, the pixels are not processed in any way.
- The resource file size cannot exceed 128 Mb.
- The automatic encoding detection by BOM (header) presence is performed for string files. If BOM is absent, the encoding is defined by the file contents. The files in the ANSI, UTF-8 and UTF-16 encodings are supported. All strings are converted to Unicode when reading data from the files.

OpenCL programs

Using the resource string variables may greatly facilitate the development of some programs. For example, you are able to write a code of an OpenCL program in a separate CL file and then include it as a string into your MQL5 program resources.

```
#resource "seascape.cl" as string cl_program
...
int context;
if((cl_program=CLProgramCreate(context,cl_program)!=INVALID_HANDLE)
  {
   //--- perform further actions with an OpenCL program
  }
```

In this example, you would have had to write the entire code as a single big string if no *cl_program* resource variable had been used.

See also

ResourceCreate(), ResourceSave(), PlaySound(), ObjectSetInteger(), ChartApplyTemplate(), File Functions

Call of Imported Functions

To import functions during the execution of a mql5-program, the client terminal uses early binding. This means that if a program has call of an imported function, the corresponding module (ex5 or dll) is loaded during the program load. MQL5 and DLL libraries are executed in the thread of a calling module.

It is not recommended to use the fully specified name of the module to be loaded like *Drive:\Directory\FileName.Ext*. The MQL5 libraries are loaded from the *terminal_dir\MQL5\Libraries* folder. If the library hasn't been found, then the client terminal performs an attempt to load it from *terminal_dir\experts* folder.

The system libraries (DLL) are loaded by the operating system rules. If the library is already loaded (for example, another Expert Advisor, and even from another client terminal, running in parallel), then it uses requests to the library already loaded. Otherwise, it performs a search in the following sequence:

1. Directory, from which the module importing dll was started. The module here is an Expert Advisor, a script, an indicator or EX5 library;
2. Directory terminal_data_directory\MQL5\Libraries (TERMINAL_DATA_PATH\MQL5\Libraries);
3. Directory, from which the MetaTrader 5 client terminal was started;
4. System directory;
5. Windows directory;
6. Current directory;
7. Directories listed in the PATH system variable.

If the DLL library uses another DLL in its work, the first one cannot be loaded in case when there is no second DLL.

Before an Expert Advisor (script, indicator) is loaded, a common list of all EX5 library modules is formed. It's going to be used both from a loaded Expert Advisor (script, indicator) and from libraries of this list. Thus the one-time loading of many times used EX5 library modules is needed. Libraries use predefined variables of the Expert Advisor (script, indicator) they were called by.

The imported library EX5 is searched for in the following sequence:

1. Directory, path to which is set relative to the directory of the Expert Advisor (script, indicator) that imports EX5;
2. Directory terminal_directory\MQL5\Libraries;
3. Directory MQL5\Libraries in the common directory of all MetaTrader 5 client terminals (Common\MQL5\Libraries).

Functions imported DLL into a mql5-program must ensure the Windows API calls agreement. To ensure such an agreement, in the source text of programs written in C or C++, use the keyword __stdcall, which is specific to the Microsoft(r) compilers. This agreement is characterized by the following:

- caller (in our case it is a mq5-program) should "see" a prototype of a function called (imported from the DLL), in order to properly combine parameters to a stack;
- caller (in our case it is a mql5-program) puts parameters to the stack in a reverse order, from right to left - in this order an imported function reads parameters passed to it;
- parameters are passed by value, except those explicitly passed by reference (in our case strings)

- an imported function cleans the stack independently by reading parameters passed to it.

When describing the prototype of an imported function, default parameters can be used.

If the corresponding library is unable to load, or there is a prohibition on the DLL use, or the imported function is not found - the Expert Advisor stops its operation with the appropriate message "Expert Advisor stopped" in the Journal (log file). In this case the Expert Advisor will not run until it is reinitialized. An Expert Advisor can be reinitialized as a result of recompilation or after the table of its properties is opened and OK is pressed.

Passing Parameters

All parameters of simple types are passed by values unless it is explicitly indicated that they are passed by reference. When a string is passed, the address of the buffer of the copied string is passed; if a string is passed by reference, the address of the buffer of this string without copying it is passed to the function imported from DLL.

Structures that contain dynamic arrays, strings, classes, other complex structures, as well as static or dynamic arrays of the enumerated objects, can't be passed as a parameter to an imported function.

When passing an array to DLL, the address of the beginning of the data buffer is always passed (irrespective of the AS_SERIES flag). A function inside a DLL knows nothing about the AS_SERIES flag, the passed array is a static array of an undefined length; an additional parameter should be used for specifying the array size.

Runtime Errors

The executing subsystem of the client terminal has an opportunity to save the error code in case it occurs during a MQL5 program run. There is a predefined variable _LastError for each executable MQL5 program.

Before starting the OnInit function, the _LastError variable is reset. In case an erroneous situation occurs during calculations or in the process of internal function calls, the _LastError variable accepts a corresponding error code. The value stored in this variable can be obtained using the GetLastError() function.

There are several critical errors in case of which a program is terminated immediately:

- division by zero
- going beyond array boundary
- using an incorrect object pointer

Testing Trading Strategies

The idea of automated trading is appealing by the fact that the trading robot can work non-stop for 24 hours a day, seven days a week. The robot does not get tired, doubtful or scared, it's is totally free from any psychological problems. It is sufficient enough to clearly formalize the trading rules and implement them in the algorithms, and the robot is ready to work tirelessly. But first, you must make sure that the following two important conditions are met:

- The Expert Advisor performs trading operations in accordance with the rules of the trading system;
- The trading strategy, implemented in the EA, demonstrates a profit on the history.

To get answers to these questions, we turn to the Strategy Tester, included in the MetaTrader 5 client terminal.

This section covers the features of program testing and optimization in the strategy tester:

- Function Limitations in the Strategy Tester
- Tick Generation Modes
- Simulation of spread
- Using real ticks during a test
- The Global Variables of the Client Terminal
- The Calculation of Indicators During Testing
- Loading History during Testing
- Multi-Currency Testing
- Simulation of Time in the Strategy Tester
- Graphical Objects in Testing
- The OnTimer() Function in the Strategy Tester
- The Sleep() Function in the Strategy Tester
- Using the Strategy Tester for Optimization Problems in Mathematical Calculations
- The Synchronization of Bars in the "Open prices only" mode
- The IndicatorRelease() function in the Tester
- Event Handling in the Tester
- Testing Agents
- The Data Exchange between the Terminal and the Agent
- Using the Shared Folder of All of the Client Terminals
- Using DLLs

Memory and disk space limits in MQL5 Cloud Network

The following limitation applies to optimizations run in the MQL5 Cloud Network: the Expert Advisor must not write to disk more than 4GB of information or use more than 4GB of RAM. If the limit is exceeded, the network agent will not be able to complete the calculation correctly, and you will not receive the result. However, you will be charged for all the time spent on the calculations.

If you need to get information from each optimization pass, send frames without writing to disk. To avoid using file operations in Expert Advisors during calculations in the MQL5 Cloud Network, you can use the following check:

```
   int handle=INVALID_HANDLE;
   bool file_operations_allowed=true;
   if(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_FORWARD))
      file_operations_allowed=false;

   if(file_operations_allowed)
     {
      ...
      handle=FileOpen(...);
      ...
     }
```

Function Limitations in the Strategy Tester

There are operation limitations for some functions in the client terminal's Strategy Tester.

The Comment(), Print() and PrintFormat() Functions

To increase performance Comment(), Print() and PrintFormat() functions are not executed when optimizing the trading robot's parameters. The exception is the use of these functions inside the OnInit() handler. This allows you to easily find the cause of errors when they occur.

The Alert(), MessageBox(), PlaySound(), SendFTP, SendMail(), SendNotification(), WebRequest() Functions

The Alert(), MessageBox(), PlaySound(), SendFTP(), SendMail(), SendNotification() and WebRequest() functions designed for interaction with the "outside world" are not executed in the Strategy Tester.

Tick Generation Modes

An Expert Advisor is a program, written in MQL5, that is run each time in response to some external event. The EA has a corresponding function (event handler) for each pre-defined event.

The NewTick event (price change) is the main event for the EA and, therefore, we need to generate a tick sequence to test the EA. There are 3 modes of tick generation implemented in the Strategy Tester of MetaTrader 5 client terminal:

- Every tick
- 1 Minute OHLC (OHLC prices with minute bars)
- Open prices only

The basic and the most detailed is the "Every tick" mode, the other two modes are the simplifications of the basic one, and will be described in comparison to the "Every tick" mode. Consider all three modes in order to understand the differences between them.

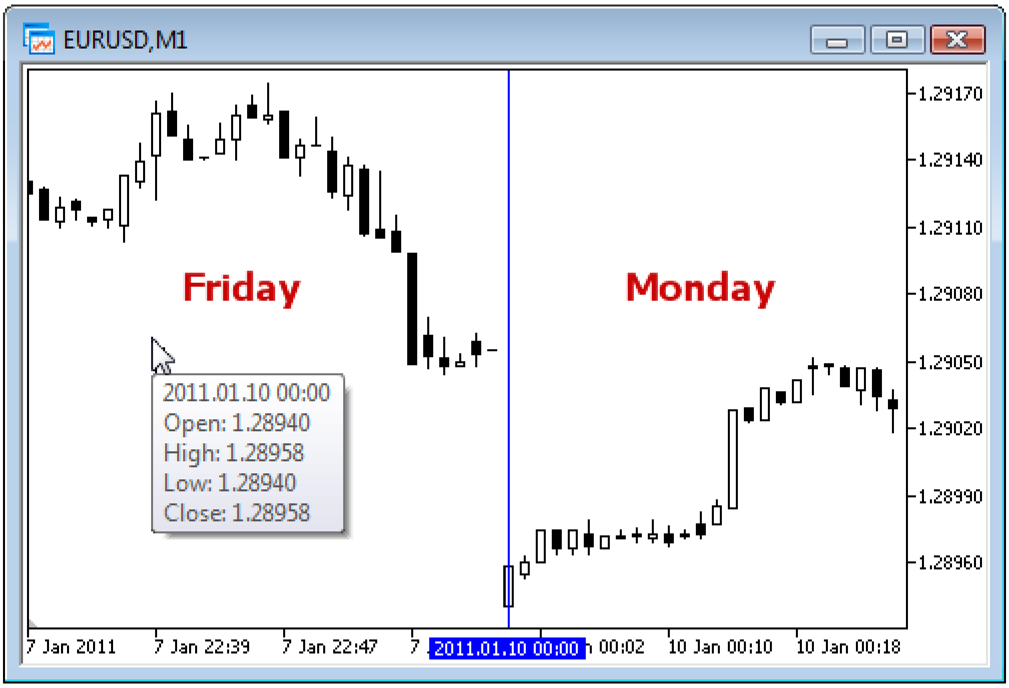
"Every Tick"

The historical quotes data for financial instruments is transferred from the trading server to the MetaTrader 5 client terminal in the form of packed minute bars. Detailed information on the occurrence of requests and the construction of the required time-frames can be obtained from the Organizing Data Access chapter of MQL5 Reference.

The minimal element of the price history is the minute bar, from which you can obtain information on the four values of the price:

- Open - the price at which the minute bar was opened;
- High - the maximum that was achieved during this minute bar;
- Low - the minimum that was achieved during this minute bar;
- Close - the closing price of the bar.

The new minute bar is not opened at the moment when the new minute begins (number of seconds becomes equal to 0), but when a tick occurs - a price change by at least one point. The figure shows the first minute bar of the new trading week, which has the opening time of 2011.01.10 00:00. The price gap between Friday and Monday, which we see on the chart, is common, since currency rates fluctuate even on weekends in response to incoming news.



2011.01.10 00:00
Open: 1.28940
High: 1.28958
Low: 1.28940
Close: 1.28958

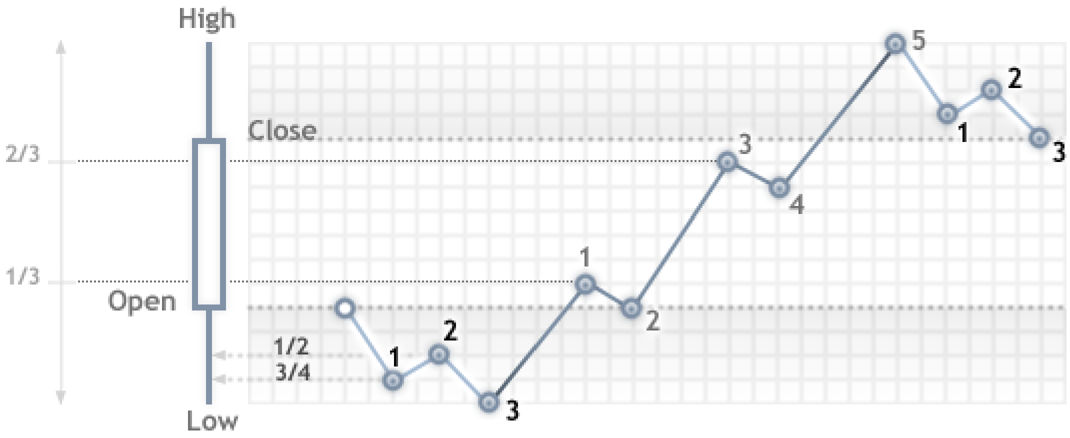
For this bar, we only know that the minute bar was opened on January 10th 2011 at 00 hours 00 minutes, but we know nothing about the seconds. It could have been opened at 00:00:12 or 00:00:36 (12 or 36 seconds after the start of a new day) or any other time within that minute. But we do know that the Open price of EURUSD was at 1.28940 at the opening time of the new minute bar.

We also don't know (accurate within a second) when we received the tick corresponding to the closing price of the considered minute bar. We known only one thing - the last Close price of the minute bar.

For this minute, the price was 1.28958. The time of the appearance of High and Low prices is also unknown, but we know that the maximum and minimum prices were on the levels of 1.28958 and 1.28940, respectively.

To test the trading strategy, we need a sequence of ticks, on which the work of the Expert Advisor will be simulated. Thus, for every minute bar, we know the **4 control points**, where the price has definitely been. If a bar has only 4 ticks, then this is enough information to perform a testing, but usually the tick volume is greater than 4.

Hence, there is a need to generate additional control points for ticks, which occurred between the Open, High, Low, and Close prices. The principle of the "Every tick" ticks generation mode is described in the The Algorithm of Ticks' Generation within the Strategy Tester of the MetaTrader 5 Terminal a figure from which is presented below.



When testing in the "Every tick" mode, the OnTick() function of the EA will be called at every control point. Each control point is a tick from a generated sequence. The EA will receive the time and price of the simulated tick, just as it would when working online.

Important: the "Every tick" testing mode is the most accurate, but at the same time, the most time consuming. For an initial testing of the majority of trading strategies, it is usually sufficient to use one of the other two testing modes.

"1 Minute OHLC"

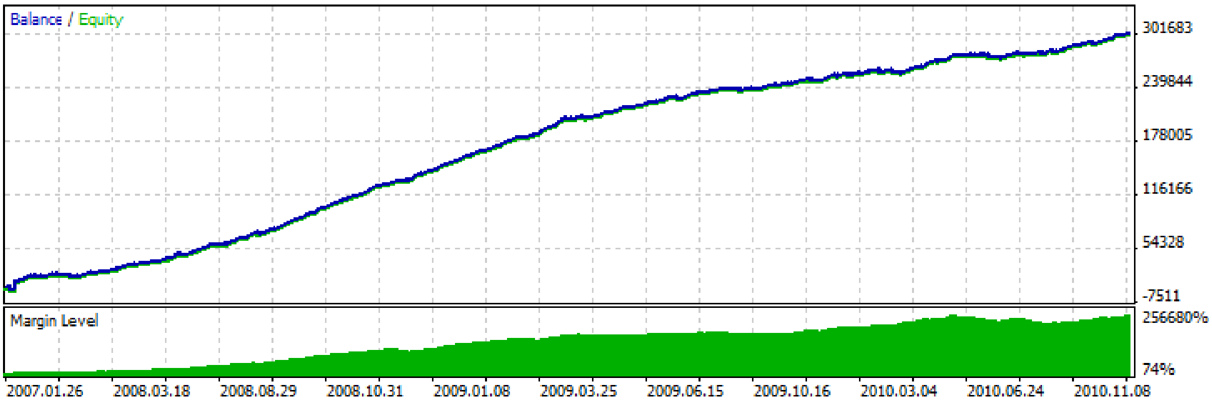
The "Every tick" mode is the most accurate of the three modes, but at the same time, is the slowest. The running of the OnTick() handler occurs at every tick, while tick volume can be quite large. For a strategy, in which the tick sequence of price movement throughout the bar, does not matter, there is a faster and rougher simulation mode - "1 minute OHLC".

In the "1 minute OHLC" mode, the tick sequence is constructed only by the **OHLC prices of the minute bars**, the number of the generated control points is significantly reduced - hence, so is the testing time. The launch of the OnTick () function is performed on all control points, which are constructed by the prices of OHLC minute bars.

The refusal to generate additional intermediate ticks between the Open, High, Low, and Close prices, leads to an appearance of rigid determinism in the development of prices, from the moment that the

Open price is determined. This makes it possible to create a "Testing Grail", which shows a nice upward graph of the testing balance.

An example of such Grail is presented in the CodeBase - Grr-al.



The figure shows a very attractive graph of this EA testing. How was it obtained? We know 4 prices for a minute bar, and we also know that the first is the Open price, and the last is the Close price. We have the High and Low prices between them, and the sequence of their occurrence is unknown, but it is known, that the High price is greater than or equal to the Open price (and the Low price is less than or equal to the Open price).

It is sufficient enough to determine the moment of receiving the Open price, and then analyze the next tick in order to determine what price we have at the moment - either the High or the Low. If the price is below the Open price, then we have a Low price and buy at this tick, the next tick will correspond to the High price, at which we will close the buy and open for sell. The next tick is the last one, this is the Close price, and we close the sale on it.

If after the price, we receive a tick with a price greater than the opening price, then the sequence of deals is reversed. Process a minute bar in this "cheat" mode, and wait for the next one.

When testing such EA on the history, everything goes smoothly, but once we launch it online, the truth begins to get revealed - the balance line remains steady, but heads downwards. To expose this trick, we simply need to run the EA in the "Every tick" mode.

> **Note:** If the test results of the EA in the rough testing modes ("1 minute OHLC" and "Open Prices only") seem too good, make sure to test it in the "Every tick" mode.

"Open Prices Only"

In this mode ticks are generated based on the OHLC prices of the timeframe selected for testing. The OnTick() function of the Expert Advisor runs only at the beginning of the bar at the Open price. Due to this feature, stop levels and pending may trigger at a price that differs from the specified one (especially when testing on higher timeframes). Instead, we have an opportunity to quickly run an evaluation test of the Expert Advisor.

W1 and MN1 periods are the exceptions in the "Open Price Only" ticks generation mode: for these timeframes ticks are generated for the OHLC prices of each day, not OHLC prices of the week or month.

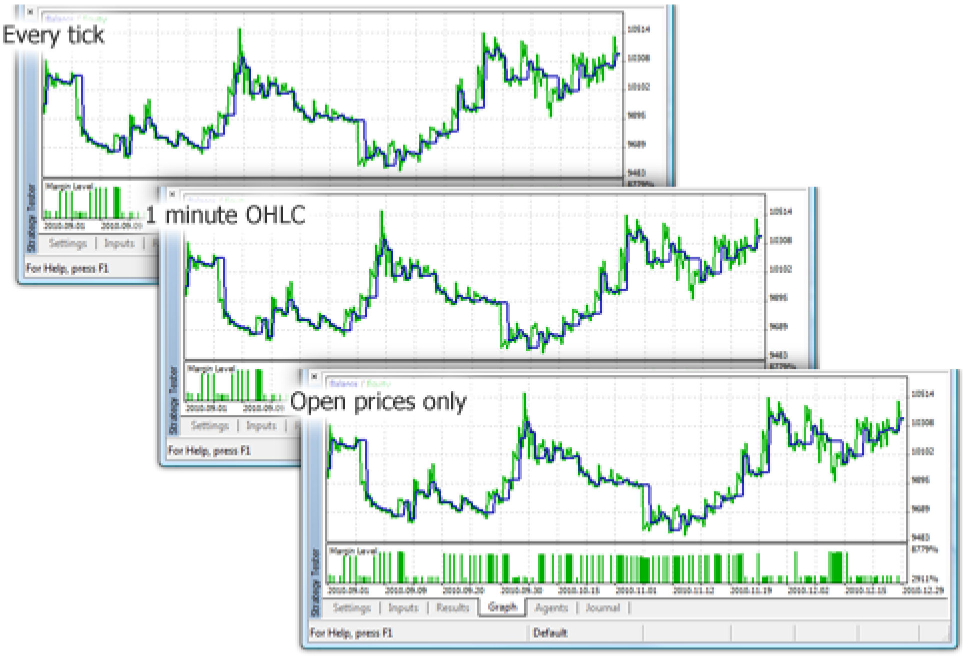
Suppose we test an Expert Advisor on EURUSD H1 in the "Open Prices Only" mode. In this case the total number of ticks (control points) will be no more than 4*number of one-hour bars within the tested interval. But the **OnTick() handler is called only at the opening of the one-hour bar**. The checks required for a correct testing occur on the rest of the ticks (that are "hidden" from the EA).

- The calculation of margin requirements;
- The triggering of Stop Loss and Take Profit levels;
- The triggering of pending orders;
- The removal of expired pending orders.

If there are no open positions or pending orders, we don't need to perform these checks on hidden ticks, and the increase of speed may be quite substantial. This "Open prices only" mode is well suited for testing strategies, which process deals only at the opening of the bar and do not use pending orders, as well as StopLoss and TakeProfit orders. For the class of such strategies, the necessary accuracy of testing is preserved.

Let's use the Moving Average Expert Advisor from the standard package as an example of an EA, which can be tested in any mode. The logic of this EA is built in such a way that all of the decisions are made at the opening of the bar, and deals are carried out immediately, without the use of pending orders.

Run a testing of the EA on EURUSD H1 on an interval from 2010.09.01 to 2010.12.31, and compare the graphs. The figure shows the balance graph from the test report for all of the three modes.



As you can see, the graphs on different testing modes are exactly the same for the Moving Average EA from the standard package.

There are some limitations on the "Open Prices Only" mode:

- You cannot use [the Random Delay execution mode](#).
- In the tested Expert Advisor, you cannot access data of the [timeframe](#) lower than that used for testing/optimization. For example, if you run testing/optimization on the H1 period, you can access

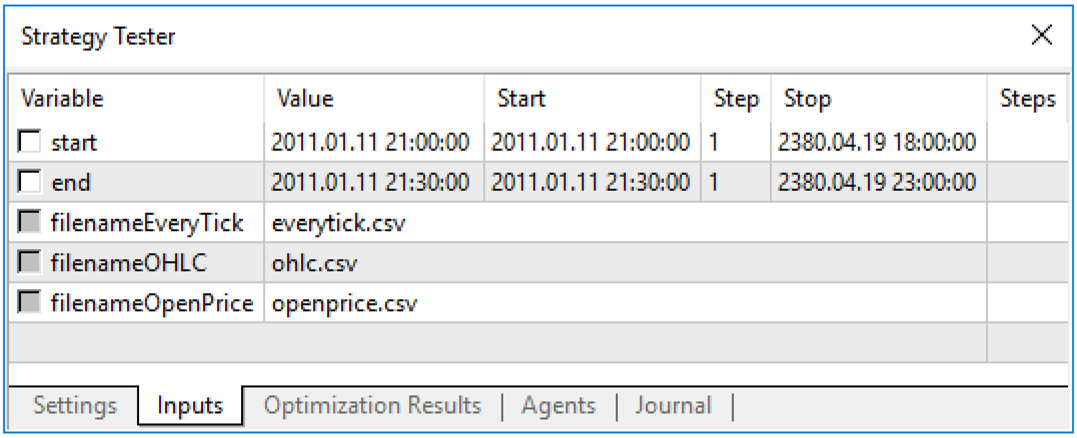
data of H2, H3, H4 etc., but not M30, M20, M10 etc. In addition, the higher timeframes that are accessed must be multiple of the testing timeframe. For example, if you run testing in M20, you cannot access data of M30, but it is possible to access H1. These limitations are connected with the impossibility to obtain data of lower or non-multiple timeframes out of the bars generated during testing/optimization.

- Limitations on accessing data of other timeframes also apply to other symbols whose data are used by the Expert Advisor. In this case the limitation for each symbol depends on the first timeframe accessed during testing/optimization. Suppose, during testing on EURUSD H1, an Expert Advisor accesses data of GBPUSD M20. In this case the Expert Advisor will be able to further use data of EURUSD H1, H2, etc., as well as GBPUSD M20, H1, H2 etc.

> **Note:** The "Open prices only" mode has the fastest testing time, but it is not suitable for all of the trading strategies. Select the desired test mode based on the characteristics of the trading system.

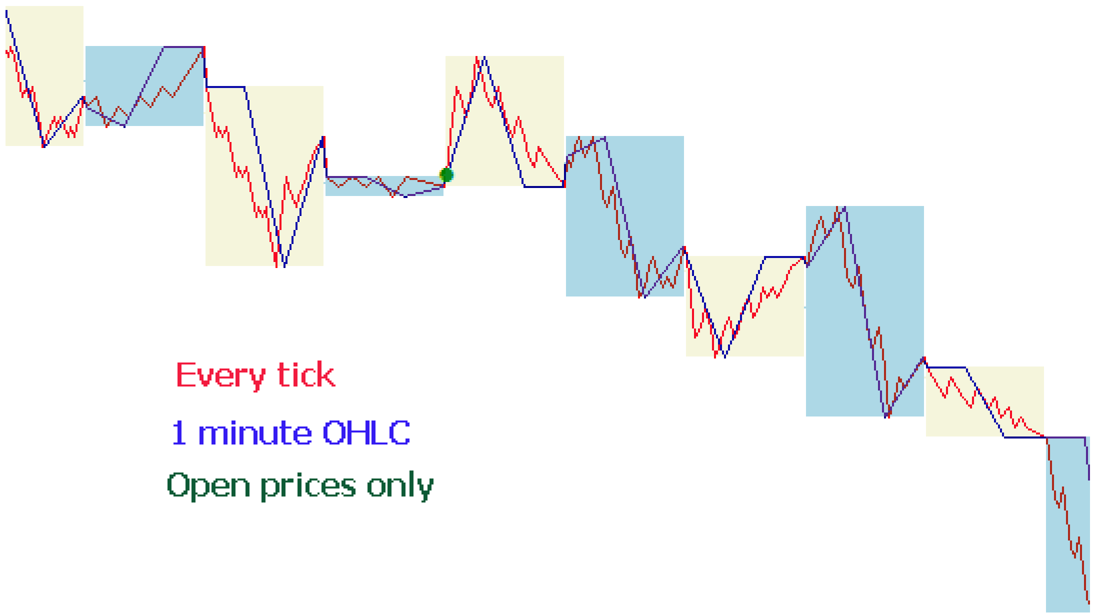
To conclude the section on the tick generation modes, let's consider a visual comparison of the different tick generation modes for EURUSD, for two M15 bars on an interval from 2011.01.11 21:00:00 - 2011.01.11 21:30:00.

The ticks were saved into different files using the WriteTicksFromTester.mq5 EA and the ending of these files names are specified in filenameEveryTick, filenameOHLC and filenameOpenPrice input-parameters.



Variable	Value	Start	Step	Stop	Steps
☐ start	2011.01.11 21:00:00	2011.01.11 21:00:00	1	2380.04.19 18:00:00	
☐ end	2011.01.11 21:30:00	2011.01.11 21:30:00	1	2380.04.19 23:00:00	
☐ filenameEveryTick	everytick.csv				
☐ filenameOHLC	ohlc.csv				
☐ filenameOpenPrice	openprice.csv				

To obtain three files with three tick sequences (for each of the following modes "Every tick", "1 minute OHLC" and "Open prices only), the EA was launched three times in the corresponding modes, in single runs. Then, the data from these three files were displayed on the chart using the TicksFromTester.mq5 indicator. The indicator code is attached to this article.



By default, all of the file operations in the MQL5 language are made within the "file sandbox", and during testing the EA has access only to its own "file sandbox". In order for the indicator and the EA to work with files from one folder during testing, we used the flag FILE_COMMON. An example of code from the EA:

```
//--- open the file
   file=FileOpen(filename,FILE_WRITE|FILE_CSV|FILE_COMMON,";");
//--- check file handle
   if(file==INVALID_HANDLE)
     {
      PrintFormat("Error in opening of file %s for writing. Error code=%d",filename,Ge
      return;
     }
   else
     {
      PrintFormat("The file will be created in %s folder",TerminalInfoString(TERMINAL_
     }
```

For reading the data in the indicator, we also used the flag FILE_COMMON. This allowed us to avoid manually transferring the necessary files from one folder to another.

```
//--- open the file
   int file=FileOpen(fname,FILE_READ|FILE_CSV|FILE_COMMON,";");
//--- check file handle
   if(file==INVALID_HANDLE)
     {
      PrintFormat("Error in open of file %s for reading. Error code=%d",fname,GetLastE
      return;
     }
   else
     {
      PrintFormat("File will be opened from %s",TerminalInfoString(TERMINAL_COMMONDAT
     }
```

Simulation of spread

The price difference between the Bid and the Ask prices is called the spread. During testing, the spread is not modeled but is taken from historical data. If the spread is less than or equal to zero in the historical data, then the last known (at the moment of generation) spread is used by testing agent.

In the Strategy Tester, the spread is always considered floating. That is, SymbolInfoInteger(symbol, SYMBOL_SPREAD_FLOAT) always returns true.

In addition, the historical data contains tick values and trading volumes. For the storage and retrieval of data we use a special MqlRates structure:

```
struct MqlRates
  {
   datetime time;          // Period start time
   double   open;          // Open price
   double   high;          // The highest price of the period
   double   low;           // The lowest price of the period
   double   close;         // Close price
   long     tick_volume;   // Tick volume
   int      spread;        // Spread
   long     real_volume;   // Trade volume
  };
```

Using real ticks during a test

Testing and optimization on real ticks are as close to real conditions as possible. Instead of generated ticks based on minute data, it is possible to use real ticks accumulated by a broker. These are ticks from exchanges and liquidity providers.

To ensure the greatest test accuracy, minute bars are also used in the real ticks mode. The bars are applied to check and correct tick data. This also allows you to avoid the divergence of charts in the tester and the client terminal.

The tester compares the tick data to the minute bar parameters: a tick should not exceed the bar's High/Low levels, also initial and final ticks should coincide with the bar's Open/Close prices. The volume is compared as well. If a mismatch is detected, all ticks corresponding to this minute bar are discarded. Generated ticks are used instead (like in the "Every tick" mode).

If a symbol history has a minute bar with no tick data for it, the tester generates ticks in the "Every tick" mode. This allows plotting a correct chart in the tester in case a broker's tick data is insufficient.

If a symbol history has no minute bar but the appropriate tick data for the minute is present, the data can be used in the tester. For example, exchange symbol pairs are formed using Last prices. If only ticks with Bid/Ask prices without the Last price arrive from the server, the bar is not generated. The tester uses these tick data since they do not contradict the minute ones.

Tick data may not coincide with minute bars for various reasons, for example because of connection losses or other failures when transmitting data from a source to the client terminal. The minute data is considered more reliable during tests.

Keep in mind the following features when testing on real ticks:

- When launching a test, the minute data on a symbol is synchronized along with the tick one.
- Ticks are stored in the symbol cache of the strategy tester. The cache size does not exceed 128 000 ticks. When new ticks arrive, the oldest data is removed from the cache. However, the CopyTicks function allows receiving ticks outside the cache (only when testing on real ticks). In that case, the data is requested from the tester tick database that is completely similar to the corresponding client terminal database. No minute bar corrections are implemented to this base. Therefore, the ticks there may differ from the ones stored in the cache.

The Global Variables of the Client Terminal

During testing, the global variables of the client terminal are also emulated, but they are not related to the current global variables of the terminal, which can be seen in the terminal using the F3 button. It means that all operations with the global variables of the terminal, during testing, take place outside of the client terminal (in the testing agent).

The Calculation of Indicators During Testing

In the real-time mode, the indicator values are calculated at every tick.

In the Strategy Tester, indicators are calculated only when they are accessed for data, i.e. when indicator buffer values are requested. The only exceptions are custom indicators with the specified #property tester_everytick_calculate. In this case, recalculation is done on each tick.

In the visual testing mode, all indicators are unconditionally recalculated when a new tick arrives in order to be correctly displayed on the visual testing chart.

The indicator is calculated once per tick. All subsequent requests for indicator data do not lead to recalculation until a new tick arrives. Therefore, if the timer is enabled in an EA via the EventSetTimer() function, the indicator data is requested from the last tick before each call of the OnTimer() handler. If the indicator has not been calculated on the last tick yet, the calculations of the indicator values are launched. If the data has already been prepared, it is provided without a new recalculation.

Thus, all indicator calculations are performed in the most resource-saving manner — if the indicator has already been calculated at a given tick, its data is provided 'as is'. No recalculation is launched.

Loading History during Testing

The history of a symbol to be tested is synchronized and loaded by the terminal from the trade server before starting the testing process. During the first time, the terminal loads all available history of a symbol in order not to request it later. Further only the new data are loaded.

A testing agent receives the history of a symbol to be tested from the client terminal right after the start of testing. If data of other instruments are used in the process of testing (for example, it is a multicurrency Expert Advisor), the testing agent requests the required history from the client terminal during the first call to such data. If historical data are available in the terminal, they are immediately passed to the testing agent. If data are not available, the terminal requests and downloads them from the server, and then passes to the testing agent.

Data of additional instruments is also required for calculating cross-rates for trade operations. For example, when testing a strategy on EURCHF with the deposit currency in USD, prior to processing the first trading operation, the testing agent requests the history data of EURUSD and USDCHF from the client terminal, though the strategy does not contain direct use call of these symbols.

Before testing a multi-currency strategy, it is recommended to download all the necessary historical data to the client terminal. This will help to avoid delays in testing/optimization associated with download of the required data. You can download history, for example, by opening the appropriate charts and scrolling them to the history beginning. An example of forced loading of history into the terminal is available in the Organizing Access to Data section of the MQL5 Reference.

Testing agents, in turn, receive history from the terminal in the packed form. During the next testing, the tester does not load history from the terminal, because the required data is available since the previous run of the tester.

- The terminal loads history from a trade server only once, the first time the agent requests the history of a symbol to be tested from the terminal. The history is loaded in a packed form to reduce the traffic.
- Ticks are not sent over the network, they are generated on testing agents.

Multi-Currency Testing

The Strategy Tester allows us to perform a testing of strategies, trading on multiple symbols. Such EAs are conventionally referred to as multi-currency Expert Advisors, since originally, in the previous platforms, testing was performed only for a single symbol. In the Strategy Tester of the MetaTrader 5 terminal, we can model trading for all of the available symbols.

The tester loads the history of the used symbols from the **client terminal** (not from the trade server!) automatically during the first call of the symbol data.

The testing agent downloads only the missing history, with a small margin to provide the necessary data on the history, for the calculation of the indicators at the starting time of testing. For the time-frames D1 and less, the minimum volume of the downloaded history is one year.

Thus, if we run a testing on an interval 2010.11.01-2010.12.01 (testing for an interval of one month) with a period of M15 (each bar is equal to 15 minutes), then the terminal will be requested the history for the instrument for the entire year of 2010. For the weekly time-frame, we will request a history of 100 bars, which is about two years (a year has 52 weeks). For testing on a monthly time-frame the agent will request the history of 8 years (12 months x 8 years = 96 months).

If there isn't necessary bars, the **starting date of testing will be automatically shifted** from past to present to provide the necessary reserve of bars before the testing.

During testing, the "Market Watch" is emulated as well, from which one can obtain information on symbols.

By default, at the beginning of testing, there is only one symbol in the "Market Watch" of the Strategy Tester - the symbol that the testing is running on. All of the necessary symbols are connected to the "Market Watch" of the Strategy Tester (not terminal!) automatically when referred to.

> Prior to starting testing of a multi-currency Expert Advisor, it is necessary to select symbols required for testing in the "Market Watch" of the terminal and load the required data. During the first call of a "foreign" symbol, its history will be automatically synchronized between the testing agent and the client terminal. A "foreign" symbol is the symbol other than that on which testing is running.

Referral to the data of an "other" symbol occurs in the following cases:

- When using the technical indicators function and IndicatorCreate() on the symbol/timeframe;
- The request to the "Market Watch" data for the other symbol:
    1. SeriesInfoInteger
    2. Bars
    3. SymbolSelect
    4. SymbolIsSynchronized
    5. SymbolInfoDouble
    6. SymbolInfoInteger
    7. SymbolInfoString
    8. SymbolInfoTick
    9. SymbolInfoSessionQuote
    10. SymbolInfoSessionTrade
    11. MarketBookAdd
    12. MarketBookGet
- Request of the time-series for a symbol/timeframe by using the following functions:
    1. CopyBuffer
    2. CopyRates
    3. CopyTime
    4. CopyOpen
    5. CopyHigh
    6. CopyLow
    7. CopyClose
    8. CopyTickVolume
    9. CopyRealVolume
    10. CopySpread

At the moment of the first call to an "other" symbol, the testing process is stopped and the history is downloaded for the symbol/timeframe, from the terminal to the testing agent. At the same time, the generation of tick sequence for this symbol is made.

An individual tick sequence is generated for each symbol, according to the selected tick generation mode. You can also request the history explicitly for the desired symbols by calling the SymbolSelect() in the OnInit() handler - the downloading of the history will be made immediately prior to the testing of the Expert Advisor.

Thus, it does not require any extra effort to perform multi-currency testing in the MetaTrader 5 client terminal. Just open the charts of the appropriate symbols in the client terminal. The history will be automatically uploaded from the trading server for all the required symbols, provided that it contains this data.

Simulation of Time in the Strategy Tester

During testing, the local time TimeLocal() is always equal to the server time TimeTradeServer(). In turn, the server time is always equal to the time corresponding to the GMT time - TimeGMT(). This way, all of these functions display the same time during testing.

The lack of a difference between the GMT, the Local, and the server time in the Strategy Tester is done deliberately in case there is no connection to the server. The test results should always be the same, regardless of whether or not there is a connection. Information about the server time is not stored locally, and is taken from the server.

Graphical Objects in Testing

During testing/optimization graphical objects are not plotted. Thus, when referring to the properties of a created object during testing/optimization, an Expert Advisor will receive zero values.

This limitation does not apply to testing in visual mode.

The OnTimer() Function in the Strategy Tester

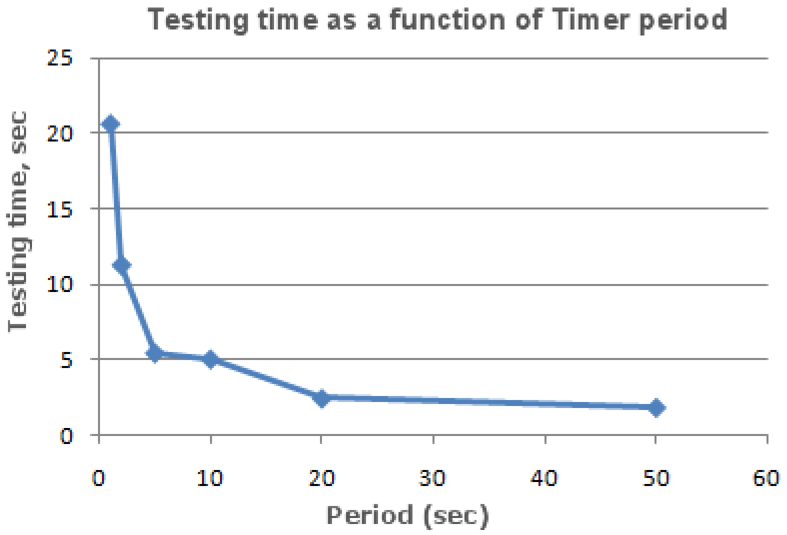
MQL5 provides the opportunity for handling timer events. The call of the OnTimer() handler is done regardless of the test mode. This means that if a test is running in the "Opening prices only" mode for the period H4, and the EA has a timer set to a call per second, then at the opening of each H4 bar, the OnTick() handler will be called one time, and the OnTimer() handler will be called 14400 times (3600 seconds * 4 hours). The amount by which the testing time of the EA will be increased depends on the logic of the EA.

To check the dependence of the testing time from the given frequency of the timer, we have created a simple EA without any trading operations.

```
//--- input parameters
input int       timer=1;              // timer value, sec
input bool      timer_switch_on=true; // timer on
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- run the timer if  timer_switch_on==true
   if(timer_switch_on)
```

```
     {
      EventSetTimer(timer);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- stop the timer
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//---
// take no actions, the body of the handler is empty
  }
//+------------------------------------------------------------------+
```

Testing time measurements were taken at different values of the timer parameter (periodicity of the Timer event). On the obtained data, we plot a testing time as function of Timer period.



It can be clearly seen that the smaller is the parameter timer, during the initialization of the EventSetTimer(Timer) function, the smaller is the period (Period) between the calls of the OnTimer() handler, and the larger is the testing time T, under the same other conditions.

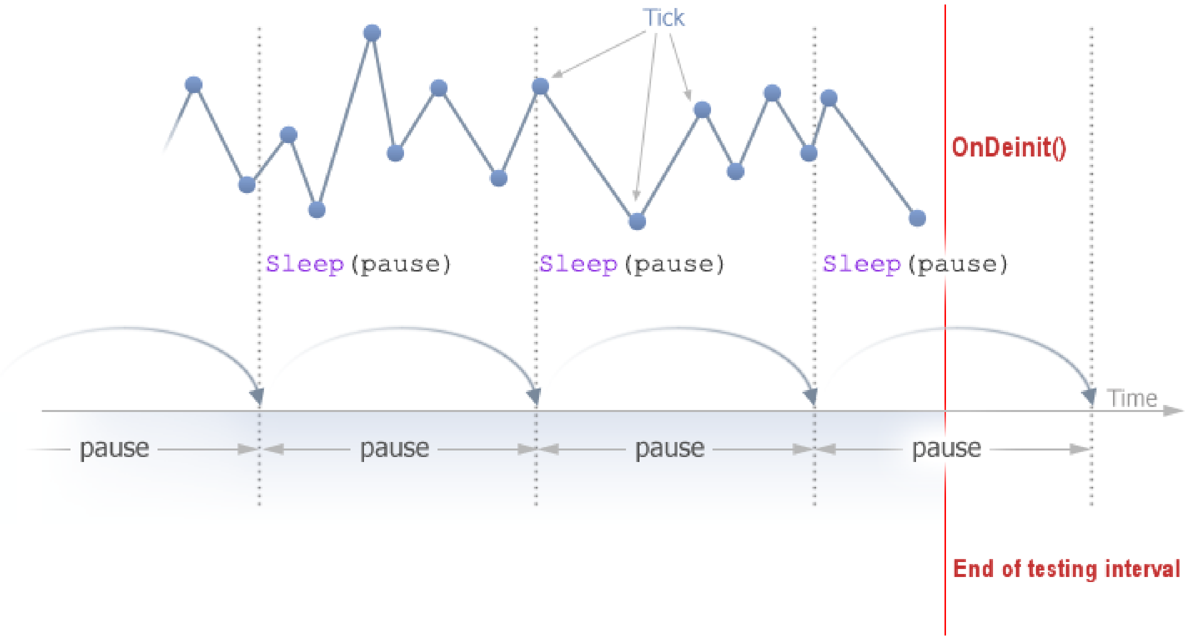
The Sleep() Function in the Strategy Tester

The Sleep() function allows the EA or script to suspend the execution of the mql5-program for a while, when working on the graph. This can be useful when requesting data, which is not ready at the time of the request and you need to wait until it is ready. A detailed example of using the Sleep() function can be found in the section Organizing Data Access.

The testing process is not lingered by the Sleep() calls.When you call the Sleep(), the generated ticks are "played" within a specified delay, which may result in the triggering of pending orders, stops, etc. After a Sleep() call, the simulated time in the Strategy Tester increases by an interval, specified in the parameter of the Sleep function.

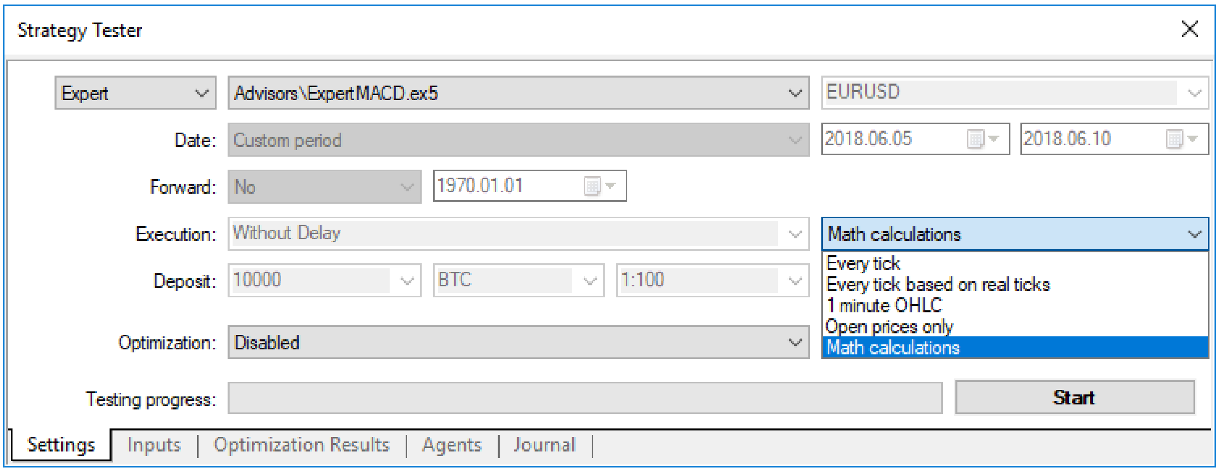
If as a result of the execution of the Sleep() function, the current time in the Strategy Tester went over the testing period, then you will receive an error "Infinite Sleep loop detected while testing". If you receive this error, the test results are not rejected, all of the computations are performed in their full volume (the number of deals, subsidence, etc.), and the results of this testing are passed on to the terminal.

The Sleep() function will not work in OnDeinit(), since after it is called, the testing time will be guaranteed to surpass the range of the testing interval.



Using the Strategy Tester for Optimization Problems in Mathematical Calculations

The tester in the MetaTrader 5 terminal can be used, not only to testing trading strategies, but also for mathematical calculations. To use it, it's necessary to select the "Math calculations" mode:



Strategy Tester

Expert: Advisors\ExpertMACD.ex5
EURUSD
Date: Custom period 2018.06.05 2018.06.10
Forward: No 1970.01.01
Execution: Without Delay
Math calculations
Deposit: 10000 BTC 1:100
Optimization: Disabled
Testing progress:

In this case, only three functions will be called: OnInit(), OnTester(), OnDeinit(). In "Math calculations" mode the Strategy Tester doesn't generate any ticks and download the history.

The Strategy Tester works in "Math calculations" mode also if you specify the starting date greater than ending date.

When using the tester to solve mathematical problems, the uploading of the history and the generation of ticks does not occur.

A typical mathematical problem for solving in the MetaTrader 5 Strategy Tester - searching for an extremum of a function with many variables.

To solve it we need to:

- The calculation of function value should be located in OnTester() function;
- The function parameters must be defined as input-variables of the Expert Advisor;

Compile the EA, open the "Strategy Tester" window. In the "Input parameters" tab, select the required input variables, and define the set of parameter values by specifying the start, stop and step values for each of the function variables.

Select the optimization type - "Slow complete algorithm" (full search of parameters space) or "Fast genetic based algorithm". For a simple search of the extremum of the function, it is better to choose a fast optimization, but if you want to calculate the values for the entire set of variables, then it is best to use the slow optimization.

Select "Math calculation" mode and using the "Start" button, run the optimization procedure. Note that during the optimization the Strategy Tester will search for the maximum values of the OnTester function. To find a local minimum, return the inverse of the computed function value from the OnTester function:

```
return(1/function_value);
```

It is necessary to check that the function_value is not equal to zero, since otherwise we can obtain a critical error of dividing by zero.

There is another way, it is more convenient and does not distort the results of optimization, it was suggested by the readers of this article:

```
return(-function_value);
```

This option does not require the checking of the function_value for being equal to zero, and the surface of the optimization results in a 3D-representation has the same shape. The only difference is that it is mirrored comparing to the original.

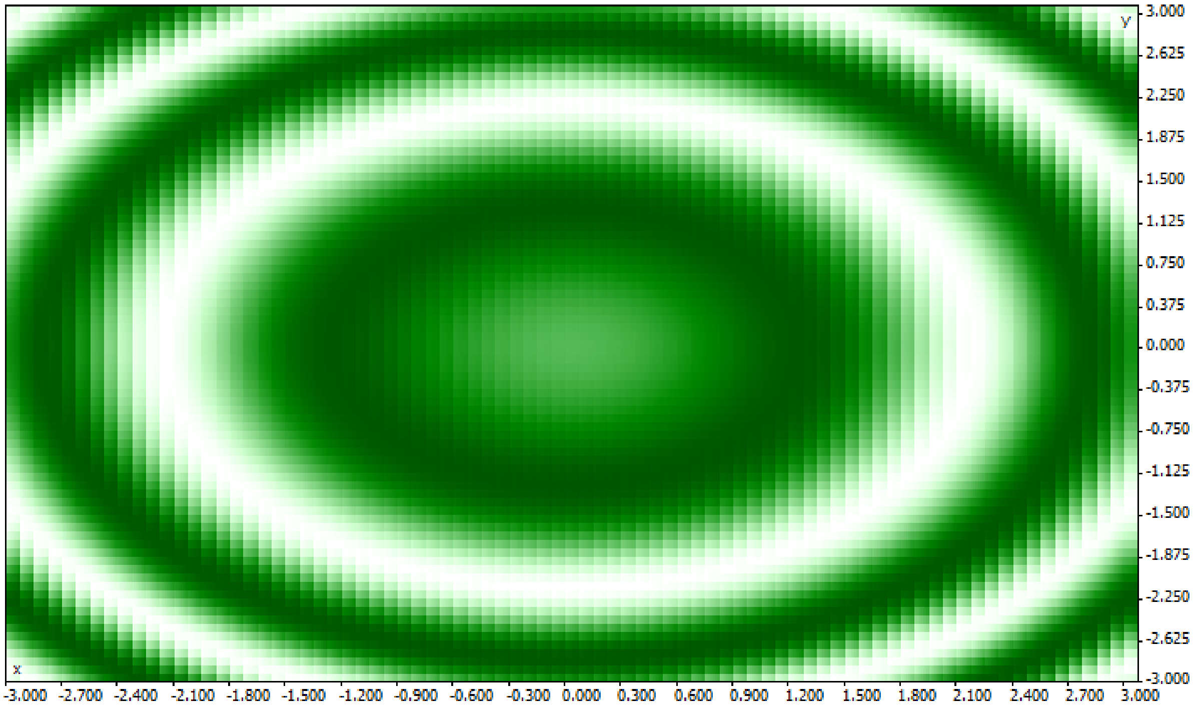
As an example, we provide the sink() function:

$$sink(x,y) = \sin\left(x^2 + y^2\right)$$

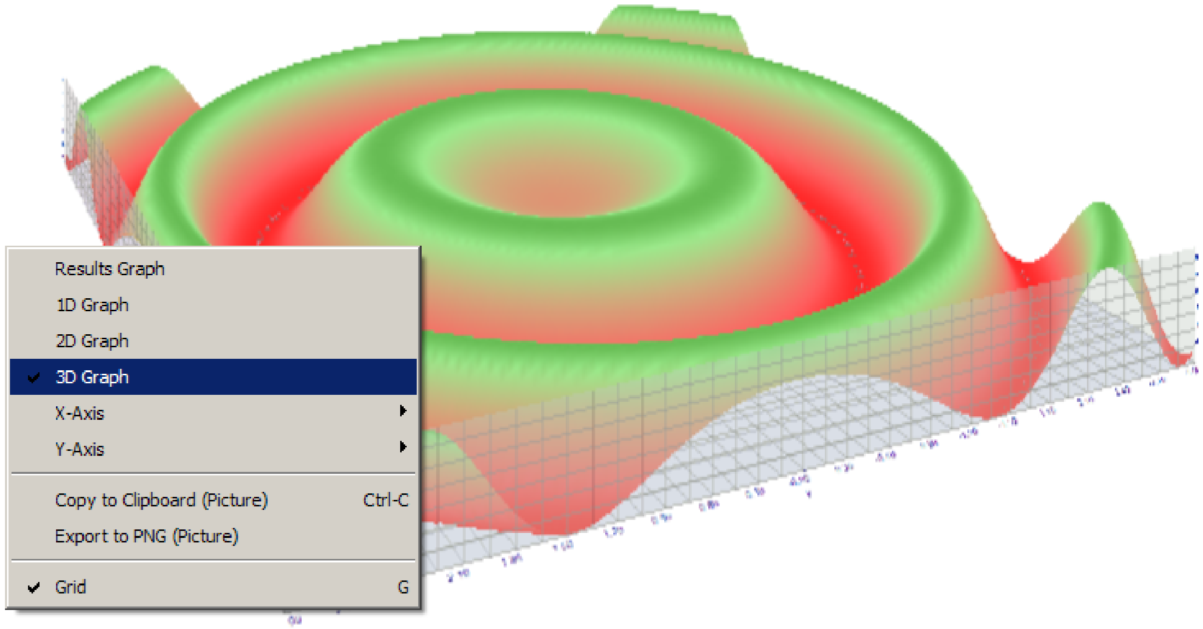
The code of the EA for finding the extremum of this function is placed into the OnTester():

```
//+------------------------------------------------------------------+
//|                                                      Sink.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input double   x=-3.0; // start=-3, step=0.05, stop=3
input double   y=-3.0; // start=-3, step=0.05, stop=3
//+------------------------------------------------------------------+
//| Tester function                                                  |
//+------------------------------------------------------------------+
double OnTester()
  {
//---
   double sink=MathSin(x*x+y*y);
//---
   return(sink);
  }
//+------------------------------------------------------------------+
```

Perform an optimization and see the optimization results in the form of a 2D graph.



The better the value is for a given pair of parameters (x, y), the more saturated the color is. As was expected from the view of the form of the sink() formula, its values forms concentric circles with a center at (0,0). One can see in the 3D-graph, that the sink() function has no single global extremum:


Results Graph
1D Graph
2D Graph
✓ 3D Graph
X-Axis
Y-Axis
Copy to Clipboard (Picture) Ctrl-C
Export to PNG (Picture)
✓ Grid G

The Synchronization of Bars in the "Open prices only" mode

The tester in the MetaTrader 5 client terminal allows us to check the so-called "multi-currency" EAs. A multi-currency EA - is an EA that trades on two or more symbols.

The testing of strategies, that are trading on multiple symbols, imposes a few additional technical requirements on the tester:

- The generation of ticks for these symbols;
- The calculation of indicator values for these symbols;
- The calculation of margin requirements for these symbols;
- Synchronization of generated tick sequences for all trading symbols.

The Strategy Tester generates and plays a tick sequence for each instrument in accordance with the selected trading mode. At the same time, a new bar for each symbol is opened, regardless of how the bar opened on another symbol. This means that when testing a multi-currency EA, a situation may occur (and often does), when for one instrument a new bar has already opened, and for the other it has not. Thus, in testing, everything happens just like in reality.

This authentic simulation of the history in the tester does not cause any problems as long as the "Every tick" and "1 minute OHLC" testing modes are used. For these modes, enough ticks are generated for one candlestick, to be able to wait until the synchronization of bars from different symbols takes place. But how do we test multi-currency strategies in the "Open prices only" mode, if the synchronization of bars on trading instruments is mandatory? In this mode, the EA is called only on one tick, which corresponds to the time of the opening of the bars.

We'll illustrate it on an example: we are testing an EA on the EURUSD, and a new H1 candlestick has been opened on EURUSD. We can easily recognize this fact - while testing in the "Open prices only" mode, the NewTick event corresponds to the moment of a bar opening on the tested period. But there is no guarantee that the new candlestick was opened on the USDJPY symbol, which is used in the EA.

Under normal circumstances, it is sufficient enough to complete the work of the OnTick() function and to check for the emergence of a new bar on USDJPY at the next tick. But when testing in the "Open prices only" mode, there will be no other tick, and so it may seem that this mode is not fit for testing multi-currency EAs. But this is not so - do not forget that the tester in MetaTrader 5 behaves just as it would in real life. You can wait until a new bar is opened on another symbols using the function Sleep()!

The code of the EA Synchronize_Bars_Use_Sleep.mq5, which shows an example of the synchronization of bars in the "Open prices only" mode:

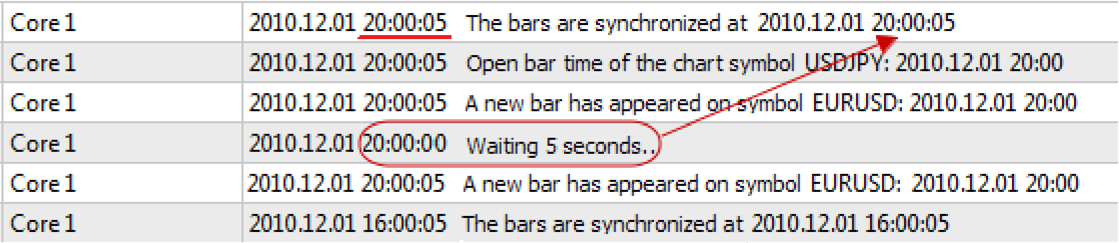
```
//+------------------------------------------------------------------+
//|                                   Synchronize_Bars_Use_Sleep.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string   other_symbol="USDJPY";
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check symbol
   if(_Symbol==other_symbol)
     {
      PrintFormat("You have to specify the other symbol in input parameters or select
      //--- forced stop testing
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- static variable, used for storage of last bar time
   static datetime last_bar_time=0;
//--- sync flag
   static bool synchonized=false;
//--- if static variable isn't initialized
   if(last_bar_time==0)
     {
      //--- it's first call, save bar time and exit
      last_bar_time=(datetime)SeriesInfoInteger(_Symbol,Period(),SERIES_LASTBAR_DATE);
      PrintFormat("The last_bar_time variable is initialized with value %s",TimeToStr
     }
//--- get open time of the last bar of chart symbol
   datetime curr_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DA
//--- if times aren't equal
   if(curr_time!=last_bar_time)
     {
      //--- save open bar time to the static variable
      last_bar_time=curr_time;
      //--- not synchronized
      synchonized=false;
      //--- print message
      PrintFormat("A new bar has appeared on symbol %s at %s",_Symbol,TimeToString(Ti
     }
//--- open time of the other symbol's bar
   datetime other_time;
//--- loop until the open time of other symbol become equal to curr_time
   while(!(curr_time==(other_time=(datetime)SeriesInfoInteger(other_symbol,Period(),S
     {
      PrintFormat("Waiting 5 seconds..");
      //--- wait 5 seconds and call SeriesInfoInteger(other_symbol,Period(),SERIES_LA
      Sleep(5000);
     }
//--- bars are synchronized
   synchonized=true;
   PrintFormat("Open bar time of the chart symbol %s: is %s",_Symbol,TimeToString(last
   PrintFormat("Open bar time of the symbol %s: is %s",other_symbol,TimeToString(othe
//--- TimeCurrent() is not useful, use TimeTradeServer()
```

```
    Print("The bars are synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS))
  }
//+------------------------------------------------------------------+
```

Notice the last line in the EA, which displays the current time when the fact of synchronization was established:

```
    Print("The bars synchronized at ",TimeToString(TimeTradeServer(),TIME_SECONDS));
```

To display the current time we used the TimeTradeServer() function rather than TimeCurrent(). The TimeCurrent() function returns the time of the last tick, which does not change after using Sleep(). Run the EA in the "Open prices only" mode, and you will see a message about the synchronization of the bars.



```
Core 1    2010.12.01 20:00:05   The bars are synchronized at 2010.12.01 20:00:05
Core 1    2010.12.01 20:00:05   Open bar time of the chart symbol USDJPY: 2010.12.01 20:00
Core 1    2010.12.01 20:00:05   A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1    2010.12.01 20:00:00   Waiting 5 seconds..
Core 1    2010.12.01 20:00:05   A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1    2010.12.01 16:00:05   The bars are synchronized at 2010.12.01 16:00:05
```

Use the TimeTradeServer() function instead of the TimeCurrent(), if you need to obtain the current server time, and not the time of the last tick arrival.

There is another way to synchronize bars - using a timer. An example of such an EA is Synchronize_Bars_Use_OnTimer.mq5, which is attached to this article.

The IndicatorRelease() function in the Tester

After completing a single testing, a chart of the instrument is automatically opened, which displays the completed deals and the indicators used in the EA. This helps to visually check the entry and exit points, and compare them with the values of the indicators.

Note: indicators, displayed on the chart, which automatically opens after the completion of the testing, are calculated anew after the completion of testing. Even if these indicators were used in the tested EA.

But in some cases, the programmer may want to hide the information on which indicators were involved in the trading algorithms. For example, the code of the EA is rented or sold as an executable file, without the provision of the source code. For this purpose, the IndicatorRelease() function is suitable.

If the terminal sets a template with the name tester.tpl in the directory/profiles/templates of the client terminal, then it will be applied to the opened chart. In its absence, the default template is applied. (default.tpl).

The IndicatorRelease() function is originally intended for releasing the calculating portion of the indicator, if it is no longer needed. This allows you to save both the memory and the CPU resources, because each tick calls for an indicator calculation. Its second purpose is to prohibit the showing of an indicator on the testing chart, after a single test run.

To prohibit the showing of the indicator on the chart after testing, call the IndicatorRelease() with the handle of the indicator in the handler OnDeinit(). The OnDeinit() function is always called after the completion and before the showing of the testing chart.

```
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   bool hidden=IndicatorRelease(handle_ind);
   if(hidden) Print("IndicatorRelease() successfully completed");
   else Print("IndicatorRelease() returned false. Error code ",GetLastError());
  }
```

In order to prohibit the showing of the indicator on the chart, after the completion of a single test, use the function IndicatorRelease() in the handler OnDeinit().

Event Handling in the Tester

The presence of the OnTick() handler in the EA is not mandatory in order for it to be subjected to testing on historical data in the MetaTrader 5 tester. It is sufficient enough for the EA ti contain at least one of the following function-handlers:

- OnTick() - Event handler of a new tick arrival;
- OnTrade() - Trading event handler;
- OnTimer() - Event handler of a signal arrival from the timer;
- OnChartEvent() - a handler for client events.

When testing in an EA, we can handle custom events using the OnChartEvent() function, but in the indicators, this function can not be called in the tester. Even if the indicator has the OnChartEvent() event handler and this indicator is used in the tested EA, the indicator itself will not receive any custom events.

During testing, an Indicator can generate custom events using the EventChartCustom() function, and the EA can process this event in the OnChartEvent().

In addition to these events, special events associated with the process of testing and optimization are generated in the strategy tester:

- Tester - this event is generated after completion of Expert Advisor testing on history data. The Tester event is handled using the OnTester() function. This function can be used only when testing Expert Advisor and is intended primarily for the calculation of a value that is used as a Custom max criterion for genetic optimization of input parameters.
- TesterInit - this event is generated during the start of optimization in the strategy tester before the very first pass. The TesterInit event is handled using the OnTesterInit() function. During the start of optimization, an Expert Advisor with this handler is automatically loaded on a separate terminal chart with the symbol and period specified in the tester, and receives the TesterInit event. The function is used to initiate an Expert Advisor before start of optimization for further processing of optimization results.

- TesterPass - this event is generated when a new data frame is received. The TesterPass event is handled using the OnTesterPass() function. An Expert Advisor with this handler is automatically loaded on a separate terminal chart with the symbol/period specified for testing, and receives the TesterPass event when a frame is received during optimization. The function is used for dynamic handling of optimization results "on the spot" without waiting for its completion. Frames are added using the FrameAdd() function, which can be called after the end of a single pass in the OnTester() handler.
- TesterDeinit - this event is generated after the end of Expert Advisor optimization in the strategy tester. The TesterDeinit event is handles using the OnTesterDeinit() function. An Expert Advisor with this handler is automatically loaded on a chart at the start of optimization, and receives TesterDeinit after its completion. The function is used for final processing of all optimization results.

Testing Agents

Testing in the MetaTrader 5 client terminal is carried out using test agents. Local agents are created and enabled automatically. The default number of local agents corresponds to the number of cores in a computer.

Each testing agent has its own copy of the global variables, which is not related to the client terminal. The terminal itself is the dispatcher, which distributes the tasks to the local and remote agents. After executing a task on the testing of an EA, with the given parameters, the agent returns the results to the terminal. With a single test, only one agent is used.

The agent stores the history, received from the terminal, in separate folders, by the name of the instrument, so the history for EURUSD is stored in a folder named EURUSD. In addition, the history of the instruments is separated by their sources. The structure for storing the history looks the following way:

```
tester_catalog\Agent-IPaddress-Port\bases\name_source\history\symbol_name
```

For example, the history for EURUSD from the server MetaQuotes-Demo can be stored in the folder tester_catalog\Agent-127.0.0.1-3000\bases\MetaQuotes-Demo\EURUSD.

A local agent, after the completion of testing, goes into a standby mode, awaiting for the next task for another 5 minutes, so as not to waste time on launching for the next call. Only after the waiting period is over, the local agent shuts down and unloads from the CPU memory.

In case of an early completion of the testing, from the user's side (the "Cancel" button), as well as with the closing of the client terminal, all local agents immediately stop their work and are unloaded from the memory.

The Data Exchange between the Terminal and the Agent

When you run a test, the client terminal prepares to send an agent a number of parameter blocks:

- Input parameters for testing (simulation mode, the interval of testing, instruments, optimization criterion, etc.)
- The list of the selected symbols in the "Market Watch"
- The specification of the testing symbol (the contract size, the allowable margins from the market for setting a StopLoss and Takeprofit, etc)

- The Expert Advisor to be tested and the values of its input parameters
- Information about additional files (libraries, indicators, data files - # property tester_ ...)

tester_indicator	string	Name of a custom indicator in the format of "indicator_name.ex5". Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a constant string. For all other cases (use of the IndicatorCreate() function or use of a non-constant string in the parameter that sets the indicator name) this property is required
tester_file	string	File name for a tester with the indication of extension, in double quotes (as a constant string). The specified file will be passed to tester. Input files to be tested, if there are necessary ones, must always be specified.
tester_library	string	Library name with the extension, in double quotes. A library can have extension dll or ex5. Libraries that require testing are defined automatically. However, if any of libraries is used by a custom indicator, this property is required

For each block of parameters, a digital fingerprint in the form of MD5-hash is created, which is sent to the agent. MD5-hash is unique for each set, its volume is many more times smaller than the amount of information on which it is calculated.

The agent receives a hash of blocks and compares them with those that it already has. If the fingerprint of the given parameter block is not present in the agent, or the received hash is different from the existing one, the agent requests this block of parameters. This reduces the traffic between the terminal and the agent.

After the testing, the agent returns to the terminal all of the results of the run, which are shown in the tabs "Test Results" and "Optimization Results": the received profit, the number of deals, the Sharpe coefficient, the result of the OnTester() function, etc.

During optimizing, the terminal hands out testing tasks to the agents in small packages, each package contains several tasks (each task means single testing with a set of input parameters). This reduces the exchange time between the terminal and the agent.

The agents never record to the hard disk the EX5-files, obtained from the terminal (EA, indicators, libraries, etc.) for security reasons, so that a computer with a running agent could not use the sent data. All other files, including DLL, are recorded in the sandbox. In remote agents you can not test EAs using DLL.

The testing results are added up by the terminal into a special cache of results (the result cache), for a quick access to them when they are needed. For each set of parameters, the terminal searches the result cache for already available results from the previous runs, in order to avoid re-runs. If the result with such a set of parameters is not found, the agent is given the task to conduct the testing.

All traffic between the terminal and the agent is encrypted.

Ticks are not sent over the network, they are generated on testing agents.

Using the Shared Folder of All of the Client Terminals

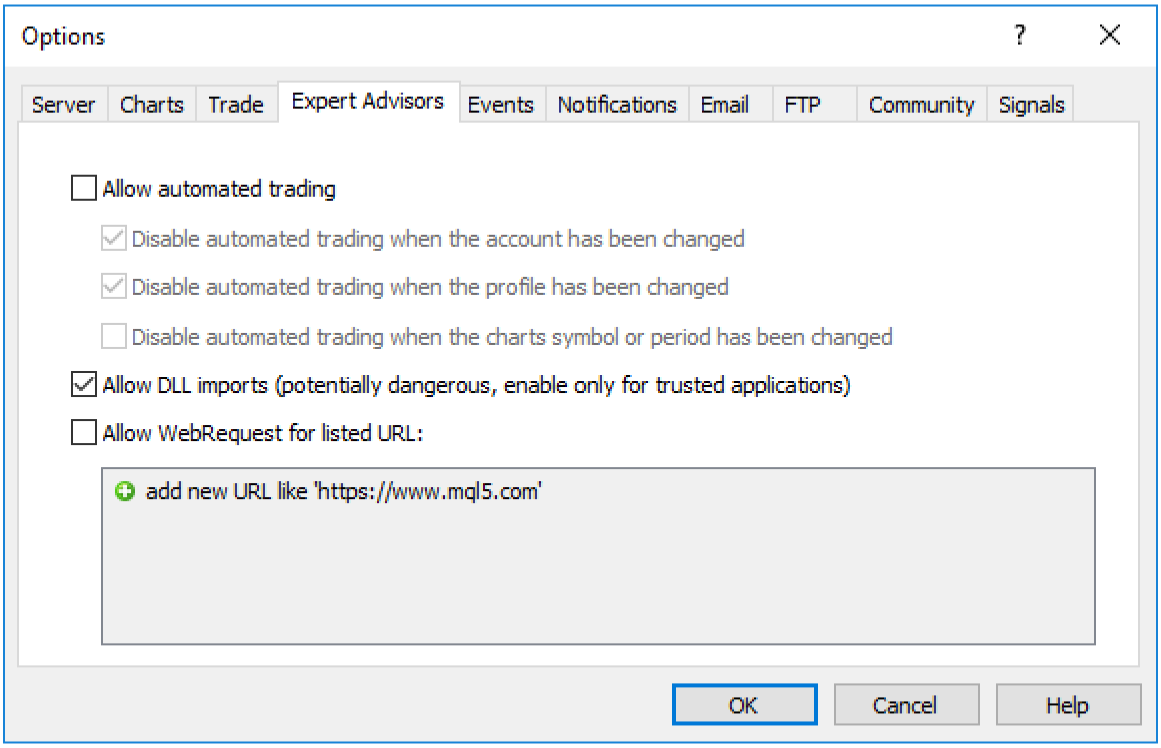
All testing agents are isolated from each other and from the client terminal: each agent has its own folder in which its logs are recorded. In addition, all file operations during the testing of the agent occur in the folder **agent_name/MQL5/Files**. However, we can implement the interaction between the local agents and the client terminal through a shared folder for all of the client terminals, if during the file opening you specify the flag FILE_COMMON:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- the shared folder for all of the client terminals
   common_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- draw out the name of this folder
   PrintFormat("Open the file in the shared folder of the client terminals %s", commo
//--- open a file in the shared folder (indicated by FILE_COMMON flag)
   handle=FileOpen(filename,FILE_WRITE|FILE_READ|FILE_COMMON);
   ... further actions
//---
   return(INIT_SUCCEEDED);
  }
```

Using DLLs

To speed up the optimization we can use not only local, but also remote agents. In this case, there are some limitations for remote agents. First of all, remote agents do not display in their logs the results of the execution of the Print() function, messages about the opening and closing of positions. A minimum of information is displayed in the log to prevent incorrectly written EAs from trashing up the computer, on which the remote agent is working, with messages.

A second limitation - the prohibition on the use of DLL when testing EAs. DLL calls are absolutely forbidden on remote agents for security reasons. On local agent, DLL calls in tested EAs are allowed only with the appropriate permission "Allow import DLL".



Options
?
Server
Charts
Trade
Expert Advisors
Events
Notifications
Email
FTP
Community
Signals
OK
Cancel
Help

☐ Allow automated trading

☑ Disable automated trading when the account has been changed

☑ Disable automated trading when the profile has been changed

☐ Disable automated trading when the charts symbol or period has been changed

☑ Allow DLL imports (potentially dangerous, enable only for trusted applications)

☐ Allow WebRequest for listed URL:

add new URL like 'https://www.mql5.com'

Note: When using 3rd party EAs (scripts, indicators) that require allowed DLL calls, you should be aware of the risks which you assume when allowing this option in the settings of the terminal. Regardless of how the EA will be used - for testing or for running on a chart.

The predefined Variables

For each executable mql5-program a set of predefined variables is supported, which reflect the state of the current price chart by the moment a mql5-program (Expert Advisor, script or custom indicator) is started.

Values of predefined variables are set by the client terminal before a mql5-program is started. Predefined variables are constant and cannot be changed from a mql5-program. As exception, there is a special variable _LastError, which can be reset to 0 by the ResetLastError function.

Variable	Value
_AppliedTo	The _AppliedTo variable allows finding out the type of data, used for indicator calculation
_Digits	Number of decimal places
_Point	Size of the current symbol point in the quote currency
_LastError	The last error code
_Period	Timeframe of the current chart
_RandomSeed	Current status of the generator of pseudo-random integers
_StopFlag	Program stop flag
_Symbol	Symbol name of the current chart
_UninitReason	Uninitialization reason code
_IsX64	The _IsX64 variable allows finding out the bit version of the terminal, in which an MQL5 application is running

Predefined variables cannot be defined in a library. A library uses such variables that are defined in program from which this library is called.

int _AppliedTo

The _AppliedTo variable allows finding out the type of data, used for indicator calculation:

Data type	Meaning	Description of data used for indicator calculation.
—	0	The indicator uses the second OnCalculate() call form - the data for calculation are not specified by a certain buffer or data array
Close	1	Close prices
Open	2	Open prices
High	3	High prices
Low	4	Low prices
Median Price (HL/2)	5	Median price = (High+Low)/2
Typical Price (HLC/3)	6	Typical price = (High+Low+Close)/3
Weighted Price (HLCC/4)	7	Weighted price = (Open+High+Low+Close)/4
Previous Indicator's Data	8	Data of the indicator, which was launched on the chart before this indicator
First Indicator's Data	9	Data of the indicator, which was launched first on the chart
Indicator handle	10+	Data of the indicator, which was passed to the iCustom() function using the indicator handle. The _AppliedTo value contains the indicator handle

Example:

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
// Getting the type of data used for indicator calculation
   Print("_AppliedTo=",_AppliedTo);
   Print(getIndicatorDataDescription(_AppliedTo));
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Description of data used for indicator calculation               |
//+------------------------------------------------------------------+
string getIndicatorDataDescription(int data_id)
  {
   string descr="";
   switch(data_id)
     {
      case(0):descr="It's first type of OnCalculate() - no data buffer";
         break;
      case(1):descr="Indicator calculates on Close price";
         break;
      case(2):descr="Indicator calculates on Open price";
         break;
      case(3):descr="Indicator calculates on High price";
         break;
      case(4):descr="Indicator calculates on Low price";
         break;
      case(5):descr="Indicator calculates on Median Price (HL/2)";
         break;
      case(6):descr="Indicator calculates on Typical Price (HLC/3)";
         break;
      case(7):descr="Indicator calculates on Weighted Price (HLCC/4)";
         break;
      case(8):descr="Indicator calculates Previous Indicator's data";
         break;
      case(9):descr="Indicator calculates on First Indicator's data";
         break;
      default: descr="Indicator calculates on data of indicator with handle="+string(
         break;
     }
//---
   return descr;
  }
```

See also

[ENUM_APPLIED_PRICE]

int _Digits

The _Digits variable stores number of digits after a decimal point, which defines the price accuracy of the symbol of the current chart.

You may also use the Digits() function.

double _Point

The _Point variable contains the point size of the current symbol in the quote currency.

You may also use the Point() function.

int _LastError

The _LastError variable contains code of the last error, that occurred during the mql5-program run. Its value can be reset to zero by ResetLastError().

To obtain the code of the last error, you may also use the GetLastError() function.

ENUM_TIMEFRAMES _Period

The _Period variable contains the value of the timeframe of the current chart.

Also you may use the Period() function.

See also

PeriodSeconds, Chart timeframes, Date and Time, Visibility of objects

_RandomSeed

Variable for storing the current state when generating pseudo-random integers. _RandomSeed changes its value when calling MathRand(). Use MathSrand() to set the required initial condition.

x random number received by MathRand() function is calculated in the following way at each call:

```
x=_RandomSeed*214013+2531011;
_RandomSeed=x;
x=(x>>16)&0x7FFF;
```

See also

MathRand(), MathSrand(), Integer types

int _StopFlag

The _StopFlag variable contains the mql5 program stop flag equal to 0 during normal operation. When the client terminal tries to stop the program, the variable is set to a value other than zero.

To check the state of the _StopFlag you may also use the IsStopped() function.

string _Symbol

The _Symbol variable contains the symbol name of the current chart.

You may also use the Symbol() function.

int _UninitReason

The _UninitReason variable contains the code of the program uninitialization reason.

Usually, this code is obtained by UninitializeReason()the function.

int _IsX64

The _IsX64 variable allows finding out the bit version of the terminal, in which an MQL5 application is running: _IsX64=0 for the 32-bit terminal and _IsX64!=0 for the 64-bit terminal.

Also, function TerminalInfoInteger(TERMINAL_X64) can be used.

Example:

```
// Checking the terminal, in which the program is running
   Print("_IsX64=",_IsX64);
   if(_IsX64)
      Print("Program ",__FILE__," is running in the 64-bit terminal");
   else
      Print("Program ",__FILE__," is running in the 32-bit terminal");
   Print("TerminalInfoInteger(TERMINAL_X64)=",TerminalInfoInteger(TERMINAL_X64));
```

See also

MQLInfoInteger, Importing functions (#import)

Common Functions

General-purpose functions not included into any specialized group are listed here.

Function	Action
Alert	Displays a message in a separate window
CheckPointer	Returns the type of the object pointer
Comment	Outputs a comment in the left top corner of the chart
CryptEncode	Transforms the data from array with the specified method
CryptDecode	Performs the inverse transformation of the data from array
DebugBreak	Program breakpoint in debugging
ExpertRemove	Stops Expert Advisor and unloads it from the chart
GetPointer	Returns the object pointer
GetTickCount	Returns the number of milliseconds that have elapsed since the system was started
GetTickCount64	Returns the number of milliseconds that have elapsed since the system was started
GetMicrosecondCount	Returns the number of microseconds that have elapsed since the start of MQL5 program
MessageBox	Creates, displays a message box and manages it
PeriodSeconds	Returns the number of seconds in the period
PlaySound	Plays a sound file
Print	Displays a message in the log
PrintFormat	Formats and prints the sets of symbols and values in a log file in accordance with a preset format
ResetLastError	Sets the value of a predetermined variable _LastError to zero
ResourceCreate	Creates an image resource based on a data set
ResourceFree	Deletes dynamically created resource (freeing the memory allocated for it)
ResourceReadImage	Reads data from the graphical resource created by ResourceCreate() function or saved in EX5 file during compilation
ResourceSave	Saves a resource into the specified file
SetUserError	Sets the predefined variable _LastError into the value equal to ERR_USER_ERROR_FIRST + user_error
SetReturnError	Sets the code that returns the terminal process when completing the operation.

Function	Action
Sleep	Suspends execution of the current Expert Advisor or script within a specified interval
TerminalClose	Commands the terminal to complete operation
TesterHideIndicators	Sets the mode of displaying/hiding indicators used in an EA
TesterStatistics	It returns the value of a specified statistic calculated based on testing results
TesterStop	Gives program operation completion command when testing
TesterDeposit	Emulates depositing funds during a test. It can be used in some money management systems
TesterWithdrawal	Emulates the operation of money withdrawal in the process of testing
TranslateKey	Returns a Unicode character by a virtual key code
ZeroMemory	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.

Alert

Displays a message in a separate window.

```
void  Alert(
   argument,      // first value
   ...            // other values
   );
```

Parameters

argument

[in] Any values separated by commas. To split the information output in several lines you can use the line feed character "\n" or "\r\n". The number of parameters can not exceed 64.

Return Value

No return value.

Note

Arrays can't be passed to the Alert() function. Arrays should be output elementwise. Data of the double type are output with 8 digits after the decimal point, data of the float type are displayed with 5 digits after the decimal point. To output the real numbers with a different precision or in a scientific format, use the DoubleToString() function.

Data of the bool type is output as "true" or "false" strings. Dates are output as YYYY.MM.DD HH:MI:SS. To display a date in another format use the TimeToString() function. Data of the color type are output either as an R,G,B string or as a color name, if the color is present in a color set.

Alert() function does not work in the Strategy Tester.

Example:

```
//--- enums
enum ENUM_INTERSECT_DIRECTION
  {
   INTERSECT_DIRECTION_NONE= 0,  // no crossing
   INTERSECT_DIRECTION_UP  = 1,  // upward crossing
   INTERSECT_DIRECTION_DOWN=-1,  // downward crossing
  };

//--- input parameters
input    uint               InpPeriod = 10;          // MA Period
input    int                InpShift  = 0;           // MA Shift
input    ENUM_MA_METHOD     InpMethod = MODE_SMA;    // MA Method
input    ENUM_APPLIED_PRICE InpPrice  = PRICE_CLOSE;    // MA Applied price

//--- global variables
int      ExtMaHandle;
int      ExtMaPeriod;
```

```
double   ExtData[2];
MqlRates ExtRates[2];

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- the period for calculating the moving average will be equal to the default value
   ExtMaPeriod=int(InpPeriod<1 ? 10 : InpPeriod);
//--- create a handle for the Moving Average indicator with the specified parameters
   ExtMaHandle=iMA(Symbol(),PERIOD_CURRENT,ExtMaPeriod,InpShift,InpMethod,InpPrice);
   ResetLastError();
   if(ExtMaHandle==INVALID_HANDLE)
     {
      PrintFormat("Failed to create iMA() handle. Error code: %d",GetLastError());
      return(INIT_FAILED);
     }

//--- get the time of the last price update
   datetime tick_time=TickTime();
//--- get moving average data and price data from the last two bars
   if(GetData(ExtMaHandle,ExtData,ExtRates) && tick_time!=0)
     {
      //--- if the price is above the MA
      if(ExtRates[1].close>ExtData[1])
        {
         //--- create a message text and display Alert
         string message=StringFormat("Bar time: %s. The price is above the moving aver
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECON
         /*
         Result:
         Alert: Bar time: 2024.02.16 18:00. The price is above the moving average at 
         */
        }
      else
        {
         //--- if the price is below the MA
         if(ExtRates[1].close<ExtData[1])
           {
            //--- create a message text and display Alert
            string message=StringFormat("Bar time: %s. The price is below the moving a
            Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SE
            /*
            Result:
            Alert: Bar time: 2024.02.16 19:00. The price is below the moving average a
            */
           }
         else
```

```
           {
            //--- create a message text and display Alert
            string message=StringFormat("Bar time: %s. The price and moving average a
            Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_S
            /*
            Result:
            Alert: Bar time: 2024.02.16 20:00. The price and moving average are equal
            */
           }
        }
     }

//--- successful
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   ResetLastError();
//--- get moving average data and price data from the last two bars
   if(!GetData(ExtMaHandle,ExtData,ExtRates))
      return;
//--- get the direction of the price crossing the moving average on the current bar
   ENUM_INTERSECT_DIRECTION intersect=GetIntersectDirection(ExtData,ExtRates);

//--- variable for saving the previous message
   static string message_prev="";

//--- if the price has crossed the moving average on the current bar upwards
   if(intersect==INTERSECT_DIRECTION_UP)
     {
      //--- get the tick time, at which the crossing occurred
      datetime tick_time=TickTime();
      if(tick_time==0)
         return;
      //--- create a message text
      string message=StringFormat("Bar time: %s. The price crossed the MA from bottom
      //--- if the previous message is not equal to the current one, display Alert wi
      if(message!=message_prev)
        {
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECO
         message_prev=message;
         /*
         Result:\
         Alert: Bar time: 2024.02.16 09:00. The price crossed the MA from bottom to t
         */
        }
     }

//--- if the price has crossed the moving average on the current bar downwards
   if(intersect==INTERSECT_DIRECTION_DOWN)
     {
      //--- get the tick time, at which the crossing occurred
      datetime tick_time=TickTime();
      if(tick_time==0)
         return;
      //--- create a message text
      string message=StringFormat("Bar time: %s. The price crossed the MA from top to
      //--- if the previous message is not equal to the current one, display Alert wit
      if(message!=message_prev)
        {
         Alert(message+" at "+TimeToString(tick_time,TIME_DATE|TIME_MINUTES|TIME_SECON
         message_prev=message;
         /*
         Result:\
         Alert: Bar time: 2024.02.16 10:00. The price crossed the MA from top to botto
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| Get price and moving average data into arrays                    |
//+------------------------------------------------------------------+
bool GetData(int handle,double &ma_data[],MqlRates &price_data[])
  {
   ResetLastError();
//--- get moving average data from the last two bars
   if(CopyBuffer(handle,0,0,2,ma_data)!=2)
     {
      PrintFormat("CopyBuffer() failed. Error code: %d",GetLastError());
      return(false);
     }
//--- get price data for the last two bars
   if(CopyRates(Symbol(),PERIOD_CURRENT,0,2,price_data)!=2)
     {
      PrintFormat("CopyRates() failed. Error code: %d",GetLastError());
      return(false);
     }

   return(true);
  }
//+------------------------------------------------------------------+
//| Return the direction of the price crossing the moving average    |
//+------------------------------------------------------------------+
ENUM_INTERSECT_DIRECTION GetIntersectDirection(double &ma_data[],MqlRates &price_data
  {
   double ma0=ma_data[1];
   double ma1=ma_data[0];
   double close0=price_data[1].close;
   double close1=price_data[0].close;

   if(close1<=ma1 && close0>ma0)
      return(INTERSECT_DIRECTION_UP);
   else
     {
      if(close1>=ma1 && close0<ma0)
         return(INTERSECT_DIRECTION_DOWN);
      else
         return(INTERSECT_DIRECTION_NONE);
     }
  }
//+------------------------------------------------------------------+
//| Return the tick time in seconds                                  |
//+------------------------------------------------------------------+
datetime TickTime()
  {
   MqlTick tick={};

   ResetLastError();
   if(!SymbolInfoTick(Symbol(),tick))
     {
      PrintFormat("SymbolInfoTick() failed. Error code: %d",GetLastError());
      return(0);
     }

   return(tick.time);
  }
```

CheckPointer

The function returns the type of the object pointer.

```
ENUM_POINTER_TYPE  CheckPointer(
   object* anyobject      // object pointer
   );
```

Parameters

anyobject

[in] Object pointer.

Return value

Returns a value from the ENUM_POINTER_TYPE enumeration.

Note

An attempt to call an incorrect pointer results in the critical termination of a program. That's why it's necessary to call the CheckPointer function before using a pointer. A pointer can be incorrect in the following cases:

- the pointer is equal to NULL;
- the object has been deleted using the delete operator.

This function can be used for checking pointer validity. A non-zero value warranties that the pointer can be used for accessing.

To quickly validate the pointer, you can also use operator "!" (example) which checks it via an implicit call of the CheckPointer function.

Example:

```
//+------------------------------------------------------------------+
//| Deletes list by deleting its elements                            |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- service pointer for working in the loop
   CItem* item;
//--- go through loop and try to delete dynamic pointers
   while(CheckPointer(m_items)!=POINTER_INVALID)
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
        {
         Print("Dynamic object ",item.Identifier()," to be deleted");
         delete (item);
        }
      else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
     }
//---
  }
```

See also

Object Pointers, Checking the Object Pointer, Object Delete Operator delete

Comment

This function outputs a comment defined by a user in the top left corner of a chart.

```
void  Comment(
   argument,      // first value
   ...            // next values
   );
```

Parameters

...

[in] Any values, separated by commas. To delimit output information into several lines, a line break symbol "\n" or "\r\n" is used. Number of parameters cannot exceed 64. Total length of the input comment (including invisible symbols) cannot exceed 2045 characters (excess symbols will be cut out during output).

Return Value

No return value

Note

Arrays can't be passed to the Comment() function. Arrays must be entered element-by-element.

Data of double type are output with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what notation will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the DoubleToString() function.

Data of bool type are output as "true" or "false" strings. Dates are shown as YYYY.MM.DD HH:MI:SS. To show dates in another format, use the TimeToString() function. Data of color type are output either as R,G,B string or as a color name, if this color is present in the color set.

Comment() function does not work during optimization in the Strategy Tester.

Example:

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
//--- Output values in three lines
   Comment(StringFormat("Show prices\nAsk = %G\nBid = %G\nSpread = %d",Ask,Bid,Spread)
  }
```

See also

ChartSetString, ChartGetString

CryptEncode

Transforms the data from array with the specified method.

```
int  CryptEncode(
   ENUM_CRYPT_METHOD    method,        // method
   const uchar&         data[],        // source array
   const uchar&         key[],         // key
   uchar&               result[]       // destination array
   );
```

Parameters

method

[in] Data transformation method. Can be one of the values of ENUM_CRYPT_METHOD enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Return Value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the error call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| ArrayToHex                                                       |
//+------------------------------------------------------------------+
string ArrayToHex(uchar &arr[],int count=-1)
  {
   string res="";
//--- check
   if(count<0 || count>ArraySize(arr))
      count=ArraySize(arr);
//--- transform to HEX string
   for(int i=0; i<count; i++)
      res+=StringFormat("%.2X",arr[i]);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   string text="The quick brown fox jumps over the lazy dog";
   string keystr="ABCDEFG";
   uchar src[],dst[],key[];
//--- prepare key
   StringToCharArray(keystr,key);
//--- copy text to source array src[]
   StringToCharArray(text,src);
//--- print initial data
   PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(s
//--- encrypt src[] with DES 56-bit key in key[]
   int res=CryptEncode(CRYPT_DES,src,key,dst);
//--- check error
   if(res>0)
     {
      //--- print encrypted data
      PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
      //--- decode dst[] to src[]
      res=CryptDecode(CRYPT_DES,dst,key,src);
      //--- check error
      if(res>0)
        {
         //--- print decoded data
         PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToSt
        }
      else
         Print("Error in CryptDecode. Error code=",GetLastError());
     }
   else
      Print("Error in CryptEncode. Error code=",GetLastError());
  }
```

See also

Array Functions, CryptDecode()

CryptDecode

Performs the inverse transformation of the data from array, tranformed by CryptEncode().

```
int  CryptEncode(
   ENUM_CRYPT_METHOD   method,        // method
   const uchar&        data[],        // source array
   const uchar&        key[],         // key
   uchar&              result[]       // destination array
   );
```

Parameters

method

[in] Data transformation method. Can be one of the values of ENUM_CRYPT_METHOD enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Return Value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the error call the GetLastError() function.

Example:

```
input string InpKey = "ABCDEFG";  // Encryption key

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string text="The quick brown fox jumps over the lazy dog";
   uchar src[],dst[],key[];

//--- prepare the encryption key
   StringToCharArray(InpKey,key);
//--- prepare the src[] source array
   StringToCharArray(text,src);
//--- display source data
   PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(
//--- encrypt the src[] array using the DES method with the key[] 56-bit key
   int res=CryptEncode(CRYPT_DES,src,key,dst);

//--- check the encryption result
   if(res>0)
     {
      //--- display encrypted data
      PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
      //--- decrypt the dst[] array data using the DES method with the key[] 56-bit ke
      res=CryptDecode(CRYPT_DES,dst,key,src);
      //--- check the result
      if(res>0)
        {
         //--- display decrypted data
         PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToSt
        }
      else
         Print("CryptDecode failed. Error: ",GetLastError());
     }
   else
      Print("CryptEncode failed. Error: ",GetLastError());
  }
//+------------------------------------------------------------------+
//| ArrayToHex                                                       |
//+------------------------------------------------------------------+
string ArrayToHex(uchar &arr[],int count=-1)
  {
   string res="";

//--- check the size
   if(count<0 || count>ArraySize(arr))
      count=ArraySize(arr);

//--- convert to hexadecimal string
   for(int i=0; i<count; i++)
      res+=StringFormat("%.2X",arr[i]);

   return(res);
  }
```

See also

[Array Functions](#), [CryptEncode()](#)

DebugBreak

It is a program breakpoint in debugging.

```
void  DebugBreak();
```

Return Value

No return value.

Note

Execution of an MQL5 program is interrupted only if a program is started in a debugging mode. The function can be used for viewing values of variables and/or for further step-by-step execution.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- compile the file using F5
//--- in debugging mode, if i == j, stop at the DebugBreak() string
   for(int i=0,j=20; i<20; i++,j--)
     {
      if(i==j)
         DebugBreak();
     }
  }
```

ExpertRemove

The function stops an Expert Advisor and unloads it from a chart.

```
void  ExpertRemove();
```

Return Value

No return value.

Note

The Expert Advisor is not stopped immediately as you call ExpertRemove(); just a flag to stop the EA operation is set. That is, any next event won't be processed, OnDeinit() will be called and the Expert Advisor will be unloaded and removed from the chart.

Calling ExpertRemove() in the strategy tester inside the OnInit() handler cancels testing on the current set of parameters. Such completion is considered an initialization error.

When calling ExpertRemove() in the strategy tester after successful initialization of an EA, a test is completed normally with the call of OnDeinit() and OnTester(). In this case, the entire trading statistics and an optimization criterion value are obtained.

Example:

```
//+------------------------------------------------------------------+
//|                                            Test_ExpertRemove.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20;// number of ticks before EA unload
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(TimeCurrent(),": " ,__FUNCTION__," reason code = ",reason);
//--- "clear" comment
   Comment("");
//---
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   static int tick_counter=0;
//---
```

```
    tick_counter++;
    Comment("\nBefore unloading expert advisor ",__FILE__," left",
            (ticks_to_close-tick_counter)," ticks");
//--- before
    if(tick_counter>=ticks_to_close)
      {
       ExpertRemove();
       Print(TimeCurrent(),": ",__FUNCTION__," expert advisor will be unloaded");
      }
    Print("tick_counter =",tick_counter);
//---
   }
//+------------------------------------------------------------------+
```

See also

Program running, Client terminal events

GetPointer

The function returns the object pointer.

```
void*  GetPointer(
   any_class anyobject      // object of any class
   );
```

Parameters

anyobject

[in] Object of any class.

Return Value

The function returns the object pointer.

Note

Only class objects have pointers. Instances of structures and simple-type variables can't have pointers. The class object not created using the new() operator, but, e.g., automatically created in the array of objects, still has a pointer. But this pointer will be of the automatic type POINTER_AUTOMATIC, therefore the delete() operator can't be applied to it. Aside from that, the type pointer doesn't differ from dynamic pointers of the POINTER_DYNAMIC type.

Since variables of structure types and simple types do not have pointers, it's prohibited to apply the GetPointer() function to them. It's also prohibited to pass the pointer as a function argument. In all these cases the compiler will notify an error.

An attempt to call an incorrect pointer causes the critical termination of a program. That's why the CheckPointer() function should be called prior to using a pointer. A pointer can be incorrect in the following cases:

- the pointer is equal to NULL;
- the object has been deleted using the delete operator.

This function can be used to check the validity of a pointer. A non-zero value guarantees, that the pointer can be used for accessing.

Example:

```
//+------------------------------------------------------------------+
//|                                              Check_GetPointer.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+------------------------------------------------------------------+
//| Class implementing the list element                              |
//+------------------------------------------------------------------+
class CItem
  {
   int               m_id;
   string            m_comment;
   CItem*            m_next;
public:
                     CItem() { m_id=0; m_comment=NULL; m_next=NULL; }
                    ~CItem() { Print("Destructor of ",m_id,
                                     (CheckPointer(GetPointer(this))==POINTER_DYNAMIC)
                                     "dynamic":"non-dynamic"); }
   void              Initialize(int id,string comm) { m_id=id; m_comment=comm; }
   void              PrintMe() { Print(__FUNCTION__,":",m_id,m_comment); }
   int               Identifier() { return(m_id); }
   CItem*            Next() {return(m_next); }
   void              Next(CItem *item) { m_next=item; }
  };
//+------------------------------------------------------------------+
//| Simplest class of the list                                       |
//+------------------------------------------------------------------+
class CMyList
  {
   CItem*            m_items;
public:
                     CMyList() { m_items=NULL; }
                    ~CMyList() { Destroy(); }
    bool             InsertToBegin(CItem* item);
    void             Destroy();
  };
//+------------------------------------------------------------------+
//| Inserts list element at the beginning                            |
//+------------------------------------------------------------------+
bool CMyList::InsertToBegin(CItem* item)
  {
   if(CheckPointer(item)==POINTER_INVALID) return(false);
//---
   item.Next(m_items);
   m_items=item;
//---
   return(true);
  }
//+------------------------------------------------------------------+
//| Deleting the list by deleting elements                           |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- service pointer to work in a loop
   CItem* item;
//--- go through the loop and try to delete dynamic pointers
   while(CheckPointer(m_items)!=POINTER_INVALID)
```

```
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
        {
         Print("Dynamyc object ",item.Identifier()," to be deleted");
         delete (item);
        }
      else Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
     }
//---
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   CMyList list;
   CItem   items[10];
   CItem*  item;
//--- create and add into the list a dynamic object pointer
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(100,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- add automatic pointers into the list
   for(int i=0; i<10; i++)
     {
      items[i].Initialize(i,"automatic");
      items[i].PrintMe();
      item=GetPointer(items[i]);
      if(CheckPointer(item)!=POINTER_INVALID)
         list.InsertToBegin(item);
     }
//--- add one more dynamic object pointer at the list beginning
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(200,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- delete all the list elements
   list.Destroy();
//--- all the list elements will be deleted after the script is over
//--- see the Experts tab in the terminal
```

```
  }
```

See also

Object Pointers, Checking the Object Pointer, Object Delete Operator delete

GetTickCount

The GetTickCount() function returns the number of milliseconds that elapsed since the system start.

```
uint  GetTickCount();
```

Return Value

Value of uint type.

Note

Counter is limited by the restrictions of the system timer. Time is stored as an unsigned integer, so it's overfilled every 49.7 days if a computer works uninterruptedly.

Example:

```
#define MAX_SIZE 40
//+------------------------------------------------------------------+
//| Script for measuring computation time of 40 Fibonacci numbers    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Remember the initial value
   uint start=GetTickCount();
//--- A variable for getting the next number in the Fibonacci series
   long fib=0;
//--- In loop calculate the specified amount of numbers from Fibonacci series
   for(int i=0;i<MAX_SIZE;i++) fib=TestFibo(i);
//--- Get the spent time in milliseconds
   uint time=GetTickCount()-start;
//--- Output a message to the Experts journal
   PrintFormat("Calculating %d first Fibonacci numbers took %d ms",MAX_SIZE,time);
//--- Script completed
   return;
  }
//+------------------------------------------------------------------+
//| Function for getting Fibonacci number by its serial number       |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- The first member of the Fibonacci series
   if(n<2) return(1);
//--- All other members are calculated by the following formula
   return(TestFibo(n-2)+TestFibo(n-1));
  }
```

See also

Date and Time, EventSetMillisecondTimer, GetTickCount64, GetMicrosecondCount

GetTickCount64

The GetTickCount64() function returns the number of milliseconds that have elapsed since the system was launched.

```
ulong  GetTickCount64();
```

Return Value

A ulong type value.

Note

The counter is limited to the accuracy of the system timer, which usually returns a result with the 10-16 millisecond precision. Unlike [GetTickCount](), which is of [uint]() type and the contents of which overflow every 49.7 days in the case of continued computer operation, GetTickCount64() can be used for the unlimited computer operation time and is not subject to overflow.

Example:

```
#define MAX_SIZE 40

//+------------------------------------------------------------------+
//| Script for measuring the time of calculating 40 Fibo numbers     |
//+------------------------------------------------------------------+
void OnStart()
  {
   long fib_array[MAX_SIZE];

//--- store the initial value
   ulong start=GetTickCount64();
//--- a loop, in which we calculate a given number of numbers from the Fibo series
   for(int i=0;i<MAX_SIZE;i++)
      fib_array[i]=TestFibo(i);
//--- get the spent time in milliseconds
   ulong time=GetTickCount64()-start;

//--- display the error message in the Experts journal
   ArrayPrint(fib_array);
   PrintFormat("Calculating the first %d Fibonacci numbers took %I64u ms",MAX_SIZE,ti
  }
//+------------------------------------------------------------------+
//| Function for obtaining a Fibo number by its serial number        |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- first member of the Fibo series
   if(n<2)
      return(1);
//--- all subsequent members are calculated using this equation
```

```
    return(TestFibo(n-2)+TestFibo(n-1));
   }
```

See also

Date and Time, EventSetMillisecondTimer, GetTickCount, GetMicrosecondCount

GetMicrosecondCount

The GetMicrosecondCount() function returns the number of microseconds that have elapsed since the start of MQL5-program.

```
ulong  GetMicrosecondCount();
```

Return Value

Value of ulong type.

Example:

```
//+------------------------------------------------------------------+
//| Test function                                                    |
//+------------------------------------------------------------------+
void Test()
  {
   int    res_int=0;
   double res_double=0;
//---
   for(int i=0;i<10000;i++)
     {
      res_int+=i*i;
      res_int++;
      res_double+=i*i;
      res_double++;
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   uint   ui=0,ui_max=0,ui_min=INT_MAX;
   ulong  ul=0,ul_max=0,ul_min=INT_MAX;
//--- number of measurements
   for(int count=0;count<1000;count++)
     {
      uint  ui_res=0;
      ulong ul_res=0;
      //---
      for(int n=0;n<2;n++)
        {
         //--- select measurement type
         if(n==0) ui=GetTickCount();
         else     ul=GetMicrosecondCount();
         //--- execute code
         Test();
         //--- add measurement result (depending on type)
         if(n==0) ui_res+=GetTickCount()-ui;
         else     ul_res+=GetMicrosecondCount()-ul;
        }
      //--- calculate minimum and maximum time for both measurements
      if(ui_min>ui_res) ui_min=ui_res;
      if(ui_max<ui_res) ui_max=ui_res;
      if(ul_min>ul_res) ul_min=ul_res;
      if(ul_max<ul_res) ul_max=ul_res;
     }
//---
   Print("GetTickCount error(msec): ",ui_max-ui_min);
   Print("GetMicrosecondCount error(msec): ",DoubleToString((ul_max-ul_min)/1000.0,2))
  }
```

See also

Date and Time, GetTickCount, GetTickCount64

MessageBox

It creates and shows a message box and manages it. A message box contains a message and header, any combination of predefined signs and command buttons.

```
int  MessageBox(
   string  text,              // message text
   string  caption=NULL,      // box header
   int     flags=0            // defines set of buttons in the box
   );
```

Parameters

text

[in] Text, containing message to output.

caption=NULL

[in] Optional text to be displayed in the box header. If the parameter is empty, Expert Advisor name is shown in the box header.

flags=0

[in] Optional flags defining appearance and behavior of a message box. Flags can be a combination of a special group of flags.

Return Value

If the function is successfully performed, the returned value is one of values of MessageBox() return codes.

Note

The function cannot be used in custom indicators since calling MessageBox() suspends the thread of execution for the whole time while waiting for the user's response. Since all indicators for each symbol are executed in a single thread, such suspension makes the operation of all charts on all timeframes for this symbol impossible.

MessageBox() function does not work in the Strategy Tester.

Example:

```
//+------------------------------------------------------------------+
//| The EA displays MessageBox window with a request for further work|
//| upon reaching specified number of series of unprofitable trades  |
//| Wait for the specified number of bars and display MessageBox     |
//| with a request to continue work.                                 |
//| To check, we need to manually open and close several positions   |
//| with a loss, since the EA does not control                       |
//| its own "positions" by magic number for simplicity.              |
//+------------------------------------------------------------------+

//--- input parameters
```

```
input uint InpMaxLossDeals      =  3;     // Max Loss deals
input uint InpInactivityNumBars =  5;     // Number of bars of advisor inactivity

//--- global variables
bool     ExtFirstStart=true;              // First launch flag
bool     ExtFlag=true;                    // Flag for allowing the EA to work
uint     ExtNumLoss;                      // Number of consecutive unprofitable trades
datetime ExtTimeLastLoss;                 // Time of the last trade to close a losing p

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- get the number of losing trades in a row and the time of the last trade to close
   ExtNumLoss=GetNumLosingTradesInRow(ExtTimeLastLoss);

   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   Comment("");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- determine how many bars have passed since the last closed losing position in the
   int bars_remaining=iBarShift(Symbol(),PERIOD_CURRENT,ExtTimeLastLoss);

//--- if this is the first launch
   if(ExtFirstStart)
     {
      //--- If a specified number of bars have already passed after a series of unpro
      if(bars_remaining>(int)InpInactivityNumBars)
         ExtFlag=true;
      ExtFirstStart=false;
     }

//--- if the EA operation flag is disabled
   if(!ExtFlag)
     {
      Comment(StringFormat("The advisor is stopped for %d bars. Num Loss positions: %u
                          (InpInactivityNumBars-bars_remaining),ExtNumLoss,TimeToStrin
      //--- if a specified number of bars have passed after a series of unprofitable p
      if(bars_remaining>(int)InpInactivityNumBars)
        {
         //--- display MessageBox window with the specified text and window title
         //--- the request window has two Yes/No buttons and an icon with a question r
         //--- the Yes button is selected by default.
         string mb_text="The specified number of bars of EA inactivity have passed.\n
         string mb_caption="Please note";
         int    mb_id=MessageBox(mb_text,mb_caption,MB_YESNO|MB_ICONQUESTION|MB_DEFBUT
         //--- if the return code from MessageBox is the Yes button pressed, set the
         if(mb_id==IDYES)
           {
            ExtFlag=true;
            return;
           }
        }
      //--- the EA operation flag is disabled, exit OnTick()
      return;
     }

//---  the EA operation flag is set - the EA works as provided by the code below
   Comment(StringFormat("The advisor is working. Num Loss positions: %u, Time last los
           ExtNumLoss,TimeToString(ExtTimeLastLoss,TIME_DATE|TIME_MINUTES|TIME_SECOND
  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
  {
//--- if the transaction type is adding a transaction to history
   if(trans.type==TRADE_TRANSACTION_DEAL_ADD)
     {
      //--- get a deal ticket and select a deal from the list by ticket
      ulong deal_ticket=trans.deal;
      if(HistoryDealSelect(deal_ticket))
        {
         //--- if this is a market exit trade, get the number of losing trades in a ro
         ENUM_DEAL_ENTRY entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(deal_ticket,DEAL
         if(entry==DEAL_ENTRY_OUT || entry==DEAL_ENTRY_INOUT || entry==DEAL_ENTRY_OUT_
            ExtNumLoss=GetNumLosingTradesInRow(ExtTimeLastLoss);
        }
     }

//--- if the number of losing trades in a row is greater than the specified value and
   if(ExtNumLoss>=InpMaxLossDeals && ExtFlag)
     {
      //--- display MessageBox window with the specified text and window title
      //--- the request window has two Yes/No buttons and an icon with an exclamation
      //--- the No button is selected by default.
      string mb_text="The number of losing trades has reached the specified maximum. T
      string mb_caption="Attention!";
      int    mb_id=MessageBox(mb_text,mb_caption,MB_YESNO|MB_ICONQUESTION|MB_DEFBUTTON
      //--- if the return code from MessageBox is the No button pressed, disable the E
      if(mb_id==IDNO)
         ExtFlag=false;
     }
  }
//+------------------------------------------------------------------+
//| Return the number of consecutive unprofitable trades             |
//| and the time of the last trade to close a losing position        |
//+------------------------------------------------------------------+
uint GetNumLosingTradesInRow(datetime &time_last_deal)
  {
//--- select the entire history
   if(!HistorySelect(0,TimeCurrent()))
      return(0);

//--- get the next trade ticket by the list of historical deals in a loop
   uint res=0;
   uint total=HistoryDealsTotal();
   for(int i=(int)total-1; i>=0; i--)
     {
      ulong deal_ticket=HistoryDealGetTicket(i);
      if(deal_ticket>0)
        {
         //--- if the deal is not for exiting the position, move on to the next one
         ENUM_DEAL_ENTRY entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(deal_ticket,DEAL
         if(entry!=DEAL_ENTRY_OUT && entry!=DEAL_ENTRY_OUT_BY && entry!=DEAL_ENTRY_INO
            continue;
         //--- if the result of closing a position has a profit, interrupt the loop
         if(!IsClosePositionWithLoss(deal_ticket))
            break;
         //--- increase the counter of consecutive loss-making trades
         res++;
         //--- write the maximum trade time into a variable (looking for the last one)
         datetime deal_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
         if(deal_time>time_last_deal)
            time_last_deal=deal_time;
        }
     }

//--- return the number of consecutive losses
   return(res);
  }
//+------------------------------------------------------------------+
//| Return the flag of closing a position with a loss                 |
//+------------------------------------------------------------------+
bool IsClosePositionWithLoss(const ulong deal_ticket)
  {
//--- get the values of the properties affecting profit from the trade
   double profit=HistoryDealGetDouble(deal_ticket,DEAL_PROFIT);
   double comission=HistoryDealGetDouble(deal_ticket,DEAL_COMMISSION);
   double swap=HistoryDealGetDouble(deal_ticket,DEAL_SWAP);
   double fee=HistoryDealGetDouble(deal_ticket,DEAL_FEE);

//--- return the flag indicating that the total value of the received properties is ne
   return(profit+comission+swap+fee<0);
  }
```

PeriodSeconds

This function returns number of seconds in a period.

```
int  PeriodSeconds(
   ENUM_TIMEFRAMES  period=PERIOD_CURRENT      // chart period
   );
```

Parameters

period=PERIOD_CURRENT

[in] Value of a chart period from the enumeration ENUM_TIMEFRAMES. If the parameter isn't specified, it returns the number of seconds of the current chart period, at which the program runs.

Return Value

Number of seconds in a selected period.

Example:

```
//--- input parameters
input ENUM_TIMEFRAMES InpPeriod1 = PERIOD_CURRENT;   // First Period
input ENUM_TIMEFRAMES InpPeriod2 = PERIOD_M1;        // Second Period
 
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of seconds in the InpPeriod1 and InpPeriod2 chart periods
   int sec1=PeriodSeconds(InpPeriod1);
   int sec2=PeriodSeconds(InpPeriod2);
//--- display the received values in the log
   PrintFormat("Seconds in period %s: %lu, in period %s: %lu",TimeframeDescription(Inp
//--- calculate how many bars of the InpPeriod2 chart period are contained in a bar wi
   int res=sec1/sec2;
   if(res==0)
      res=1;
//--- display the obtained value in the log
   PrintFormat("One bar %s contains %d bars %s",TimeframeDescription(InpPeriod1),res,T
   /*
   Result:
   Seconds in period M5: 300, in period M1: 60
   One bar M5 contains 5 bars M1
   */
  }
//+------------------------------------------------------------------+
//| Return the timeframe name                                        |
//+------------------------------------------------------------------+
string TimeframeDescription(const ENUM_TIMEFRAMES period)
  {
   return(StringSubstr(EnumToString(period==PERIOD_CURRENT ? Period() : period), 7));
  }
```

See also

[_Period](), [Chart timeframes](), [Date and Time](), [Visibility of objects]()

PlaySound

It plays a sound file.

```
bool  PlaySound(
   string  filename      // file name
   );
```

Parameters

filename

[in] Path to a sound file. If filename=NULL, the playback is stopped.

Return Value

true - if the file is found, otherwise - false.

Note

The file must be located in terminal_directory\Sounds or its sub-directory. Only WAV files are played.

Call of PlaySound() with NULL parameter stops playback.

PlaySound() function does not work in the Strategy Tester.

Example:

```
#include <Trade\Trade.mqh>
#define MAGIC (123)

//--- input parameters
input string          InpFileNameOK  = "ok.wav";             // success audio file
input string          InpFileNameErr = "timeout.wav";        // error audio file
input ENUM_ORDER_TYPE InpOrderType   = ORDER_TYPE_BUY_LIMIT;  // order type
input double          InpLots        = 0.1;                  // lots

//--- global variables
CTrade ExtTrade;

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the magic and order type by execution according to the symbol settings
   ExtTrade.SetExpertMagicNumber(MAGIC);
   ExtTrade.SetTypeFillingBySymbol(Symbol());
//--- call the function of placing an order or opening a position with sound playback
   OrderSendWithAudio();
  }
```

```
//+------------------------------------------------------------------+
//| The function places an order or opens a position                 |
//| and plays the sound of success or error                          |
//+------------------------------------------------------------------+
void OrderSendWithAudio(void)
  {
   bool     res=true;
   MqlTick tick= {};

   ResetLastError();
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      PlaySound(InpFileNameErr);
      return;
     }
//--- send a request to the server
   switch(InpOrderType)
     {
      case ORDER_TYPE_BUY :
         res=ExtTrade.Buy(InpLots);
         break;
      case ORDER_TYPE_BUY_LIMIT :
         res=ExtTrade.BuyLimit(InpLots,NormalizeDouble(tick.ask-100*Point(),Digits()))
         break;
      case ORDER_TYPE_BUY_STOP :
         res=ExtTrade.BuyStop(InpLots,NormalizeDouble(tick.ask+100*Point(),Digits()));
         break;
      case ORDER_TYPE_SELL :
         res=ExtTrade.Sell(InpLots);
         break;
      case ORDER_TYPE_SELL_LIMIT :
         res=ExtTrade.SellLimit(InpLots,NormalizeDouble(tick.bid+100*Point(),Digits())
         break;
      case ORDER_TYPE_SELL_STOP :
         res=ExtTrade.SellStop(InpLots,NormalizeDouble(tick.bid-100*Point(),Digits()))
         break;
      default :
         res=false;
     }
   if(!res)
      Print("Error ",GetLastError());
   Print(ExtTrade.ResultRetcodeDescription());

//--- if the request is accepted, play the ok.wav sound
   if(ExtTrade.ResultRetcode()==TRADE_RETCODE_DONE)
      PlaySound(InpFileNameOK);
   else
      PlaySound(InpFileNameErr);
  }
```

See also

Resources

Print

It enters a message in the Expert Advisor log. Parameters can be of any type.

```
void  Print(
   argument,      // first value
   ...            // next values
   );
```

Parameters

...

[in] Any values separated by commas. The number of parameters cannot exceed 64.

Note

Arrays cannot be passed to the Print() function. Arrays must be input element-by-element.

Data of double type are shown with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what entry will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the PrintFormat() function.

Data of bool type are output as "true" or "false" lines. Dates are shown as YYYY.MM.DD HH:MI:SS. To show data in another format, use TimeToString(). Data of color type are returned either as R,G,B line or as a color name, if this color is present in the color set.

Print() function does not work during optimization in the Strategy Tester.

Example:

```
void OnStart()
  {
//--- Output DBL_MAX using Print(), this is equivalent to PrintFormat(%%.16G,DBL_MAX)
   Print("---- how DBL_MAX looks like -----");
   Print("Print(DBL_MAX)=",DBL_MAX);
//--- Now output a DBL_MAX number using PrintFormat()
   PrintFormat("PrintFormat(%%.16G,DBL_MAX)=%.16G",DBL_MAX);
//--- Output to the Experts journal
// Print(DBL_MAX)=1.797693134862316e+308
// PrintFormat(%.16G,DBL_MAX)=1.797693134862316E+308

//--- See how float is output
   float c=(float)M_PI; // We should explicitly cast to the target type
   Print("c=",c, "    Pi=",M_PI, "    (float)M_PI=",(float)M_PI);
// c=3.14159    Pi=3.141592653589793    (float)M_PI=3.14159

//--- Show what can happen with arithmetic operations with real types
   double a=7,b=200;
   Print("---- Before arithmetic operations");
   Print("a=",a,"   b=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Divide a by b (7/200)
```

```
   a=a/b;
//--- Now emulate restoring a value in the b variable
   b=7.0/a; // It is expected that b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 - but it diffe
//--- Output the newly calculated value of b
   Print("----- After arithmetic operations");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.9999999999999716 (see that b is no more equal to 2

//--- Create a very small value epsilon=1E-013
   double epsilon=1e-13;
   Print("---- Create a very small value");
   Print("epsilon=",epsilon); // Get epsilon=1E-013
//--- Now subtract epsilon from b and again output the value to the Experts journal
   b=b-epsilon;
//--- Use two ways
   Print("---- After subtracting epsilon from the b variable");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=199.9999999999999  (now the value of b after subtracting epsilon cannot b
// Print(DoubleToString(b,16))=199.9999999999998578
//    (now the value of b after subtracting epsilon cannot be rounded to 200)
  }
```

See also

DoubleToString, StringFormat

PrintFormat

It formats and enters sets of symbols and values in the Expert Advisor log in accordance with a preset format.

```
void  PrintFormat(
   string format_string,    // format string
   ...                      // values of simple types
   );
```

Parameters

format_string

[in] A format string consists of simple symbols, and if the format string is followed by arguments, it also contains format specifications.

...

[in] Any values of simple types separated by commas. Total number of parameters can't exceed 64 including the format string.

Return Value

String.

Note

PrintFormat() function does not work during optimization in the Strategy Tester.

The number, order and type of parameters must exactly match the set of qualifiers, otherwise the print result is undefined. Instead of PrintFormat() you can use printf().

If the format string is followed by parameters, this string must contain format specifications that denote output format of these parameters. Specification of format always starts with the percent sign (%).

A format string is read from left to right. When the first format specification is met (if there is any), the value of the first parameter after the format string is transformed and output according to the preset specification. The second format specification calls transformation and output of the second parameter, and so on till the format string end.

The format specification has the following form:

%[flags][width][.precision][{h | l | ll | I32 | I64}]type

Each field of the format specification is either a simple symbol, or a number denoting a simple format option. The simplest format specification contains only the percent sign (%) and a symbol defining the type of the output parameter (for example, %s). If you need to output the percent sign in the format string, use the format specification %%.

flags

Flag	Description	Default Behavior
- (minus)	Left justification within the set width	Right justification
+ (plus)	Output of the + or - sign for values of sign types	The sign is shown only if the value is negative
0 (zero)	Zeroes are added before an output value within the preset width. If **0** flag is specified with an integer format (**i**, **u**, **x**, **X**, **o**, **d**) and accuracy specification is set (for example, %04.d), then 0 is ignored.	Nothing is added
space	A space is shown before an output value, if it is a sign and positive value	Spaces aren't inserted
#	If used together with the format **o**, **x** or **X**, then before the output value 0, 0x or 0X is added respectively.	Nothing is added
	If used together with the format **e**, **E**, **a** or **A**, value is always shown with a decimal point.	Decimal point is shown only if there is a non-zero fractional part.
	If used together with the format **g** or **G**, flag defines presence of a decimal point in the output value and prevents the cutting off of leading zeroes. Flag # is ignored when used together with formats **c**, **d**, **i**, **u**, **s**.	Decimal point is shown only if there is a non-zero fractional part. Leading zeroes are cut off.

width

A non-negative decimal number that sets the minimal number of output symbols of the formatted value. If the number of output symbols is less than the specified width, the corresponding number of spaces is added from the left or right depending on the alignment (flag –). If there is flag zero (0), the corresponding number of zeroes is added before the output value. If the number of output symbols is greater than the specified width, the output value is never cut off.

If an asterisk (*) is specified as width, value of int type must be indicated in the corresponding place of the list of passed parameters. It will be used for specifying width of the output value.

precision

A non-negative decimal number that sets the output precision - number of digits after a decimal point. As distinct from width specification, precision specification can cut off the part of fractional type with or without rounding.

The use of precision specification is different for different format types.

Types	Description	Default Behavior
a, A	Precision specification sets the number of digits after a decimal point.	Default precision - 6.
c, C	Not used	
d, i, u, o, x, X	Sets minimal number of output digits. If number of digits in a corresponding parameter is less than this precision, zeroes are added to the left of the output value. The output value isn't cut off, if the number of output digits is larger than the specified precision.	Default precision - 1.
e, E, f	Sets number of output digits after a decimal point. The last digit is rounded off.	Default precision - 6. If set precision is 0 or decimal part is absent, the decimal point is not shown.
g, G	Sets maximal number of meaningful numbers.	6 meaningful numbers are output.
s	Sets number of output symbols of a string. If the string length exceeds the precision, the string is cut off.	The whole string is output.

```
PrintFormat("1. %s", _Symbol);
PrintFormat("2. %.3s", _Symbol);
int length=4;
PrintFormat("3. %.*s", length, _Symbol);
/*
1. EURUSD
2. EUR
3. EURU
/
```

h | l | ll | I32 | I64

Specification of data sizes, passed as a parameter.

Parameter Type	Used Prefix	Joint Specifier of Type
int	l (lower case L)	d, i, o, x, or X
uint	l (lower case L)	o, u, x, or X
long	ll (two lower case L)	d, i, o, x, or X

Parameter Type	Used Prefix	Joint Specifier of Type
short	h	d, i, o, x, or X
ushort	h	o, u, x, or X
int	I32	d, i, o, x, or X
uint	I32	o, u, x, or X
long	I64	d, i, o, x, or X
ulong	I64	o, u, x, or X

type

Type specifier is the only obligatory field for formatted output.

Symbol	Type	Output Format
c	int	Symbol of short type (Unicode)
C	int	Symbol of char type (ANSI)
d	int	Signed decimal integer
i	int	Signed decimal integer
o	int	Unsigned octal integer
u	int	Unsigned decimal integer
x	int	Unsigned hexadecimal integer, using "abcdef"
X	int	Unsigned hexadecimal integer, using "ABCDEF"
e	double	A real value in the format [-] d.dddde[sign] ddd, where d - one decimal digit, dddd - one or more decimal digits, ddd - a three-digit number that determines the size of the exponent, sign - plus or minus
E	double	Similar to the format of e, except that the sign of exponent is output by upper case letter (**E** instead of e)
f	double	A real value in the format [-] dddd.dddd, where dddd - one or more decimal digits. Number of displayed digits before the decimal point depends on the size of number value. Number of digits after the decimal point depends on the required accuracy.
g	double	A real value output in **f** or **e** format depending on what output is more compact.
G	double	A real value output in **F** or **E** format depending on what output is more compact.

Symbol	Type	Output Format
a	double	A real number in format [-]0xh.hhhh **p**±dd, where h.hhhh – mantissa in the form of hexadecimal digits, using "abcdef", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
A	double	A real number in format [-]0xh.hhhh **P**±dd, where h.hhhh – mantissa in the form of hexadecimal digits, using "ABCDEF", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
s	string	String output

Instead of PrintFormat() you can use printf().

Example:

```
void OnStart()
  {
//--- trade server name
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- account number
   int login=(int)AccountInfoInteger(ACCOUNT_LOGIN);
//--- long value output
   long leverage=AccountInfoInteger(ACCOUNT_LEVERAGE);
   PrintFormat("%s %d: leverage = 1:%I64d",
               server,login,leverage);
//--- account currency
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- double value output with 2 digits after the decimal point
   double equity=AccountInfoDouble(ACCOUNT_EQUITY);
   PrintFormat("%s %d: account equity = %.2f %s",
               server,login,equity,currency);
//--- double value output with mandatory output of the +/- sign
   double profit=AccountInfoDouble(ACCOUNT_PROFIT);
   PrintFormat("%s %d: current result for open positions = %+.2f %s",
               server,login,profit,currency);
//--- double value output with variable number of digits after the decimal point
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   string format_string=StringFormat("%%s: point value  = %%.%df",_Digits);
   PrintFormat(format_string,_Symbol,point_value);
//--- int value output
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   PrintFormat("%s: current spread in points = %d ",
               _Symbol,spread);
//--- double value output in the scientific (floating point) format with 17 meaningful
   PrintFormat("DBL_MAX = %.17e",DBL_MAX);
//--- double value output in the scientific (floating point) format with 17 meaningful
   PrintFormat("EMPTY_VALUE = %.17e",EMPTY_VALUE);
//--- output using PrintFormat() with default accuracy
   PrintFormat("PrintFormat(EMPTY_VALUE) = %e",EMPTY_VALUE);
//--- simple output using Print()
   Print("Print(EMPTY_VALUE) = ",EMPTY_VALUE);
/* execution result
   MetaQuotes-Demo 1889998: leverage = 1:100
   MetaQuotes-Demo 1889998: account equity = 22139.86 USD
   MetaQuotes-Demo 1889998: current result for open positions = +174.00 USD
   EURUSD: point value  = 0.00001
   EURUSD: current spread in points = 12
   DBL_MAX = 1.79769313486231570e+308
   EMPTY_VALUE = 1.79769313486231570e+308
   PrintFormat(EMPTY_VALUE) = 1.797693e+308
   Print(EMPTY_VALUE) = 1.797693134862316e+308
*/
  }
```

See also

[StringFormat](), [DoubleToString](), [Real types (double, float)]()

ResetLastError

Sets the value of the predefined variable _LastError into zero.

```
void  ResetLastError();
```

Return Value

No return value.

Note

It should be noted that the GetLastError() function doesn't zero the _LastError variable. Usually the ResetLastError() function is called before calling a function, after which an error appearance is checked.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- reset the last error code before calling the function,
//--- otherwise, GetLastError() may return the previous error code
   long lres=SymbolInfoInteger("123456",SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
   lres=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
   ResetLastError();
   lres=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   PrintFormat("lres=%d  error=%d",lres,GetLastError());
  }
```

ResourceCreate

Creates an image resource based on a data set. There are two variants of the function:

Creating a resource based on a file

```
bool  ResourceCreate(
   const string      resource_name,       // Resource name
   const string      path                 // A relative path to the file
   );
```

Creating a resource based on the array of pixels

```
bool  ResourceCreate(
   const string      resource_name,       // Resource name
   const uint&       data[],              // Data set as an array
   uint              img_width,           // The width of the image resource
   uint              img_height,          // The height of the image resource
   uint              data_xoffset,        // The horizontal rightward offset of the up
   uint              data_yoffset,        // The vertical downward offset of the upper
   uint              data_width,          // The total width of the image based on the
   ENUM_COLOR_FORMAT color_format         // Color processing method
   );
```

Parameters

`resource_name`

[in] Resource name.

`data[][]`

[in] A one-dimensional or two-dimensional array for creating a complete image.

`img_width`

[in] The width of the rectangular image area in pixels to be placed in the resource in the form of an image. It cannot be greater than the `data_width` value.

`img_height`

[in] The height of the rectangular image area in pixels to be placed in the resource in the form of an image.

`data_xoffset`

[in] The horizontal rightward offset of the rectangular area of the image.

`data_yoffset`

[in] The vertical downward offset of the rectangular area of the image.

`data_width`

[in] Required only for one-dimensional arrays. It denotes the full width of the image from the data set. If *data_width*=0, it is assumed to be equal to *img_width*. For two-dimensional arrays the parameter is ignored and is assumed to be equal to the second dimension of the *data[]* array.

`color_format`

[in] Color processing method, from a value from the ENUM_COLOR_FORMAT enumeration.

Return Value

Returns true if successful, otherwise false. To get information about the error call the GetLastError() function. The following errors may occur:

- 4015 – ERR_RESOURCE_NAME_DUPLICATED (identical names of the dynamic and the static resource)
- 4016 – ERR_RESOURCE_NOT_FOUND (the resource is not found)
- 4017 – ERR_RESOURCE_UNSUPPORTED_TYPE (this type of resource is not supported)
- 4018 – ERR_RESOURCE_NAME_IS_TOO_LONG (the name of the resource is too long)

Note

If the second version of the function is called for creating the same resource with different width, height and shift parameters, it does not create a new resource, but simply updates the existing one.

The first version of the function is used for uploading images and sounds from files, and the second version is used only for the dynamic creation of images.

Images must be in the BMP format with a color depth of 24 or 32 bits. Sounds can only be in the WAV format. The size of the resource should not exceed 16 Mb.

ENUM_COLOR_FORMAT

Identifier	Description
COLOR_FORMAT_XRGB_NOALPHA	The component of the alpha channel is ignored
COLOR_FORMAT_ARGB_RAW	Color components are not handled by the terminal (must be correctly set by the user)
COLOR_FORMAT_ARGB_NORMALIZE	Color components are handled by the terminal

Example:

```
//+------------------------------------------------------------------+
//| Update graphical resource data                                   |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- leave if zero dimensions are passed
   if(width==0 || height==0)
      return;
//--- Update resource data and redraw the chart
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- declare the parameters of the graphical resource
   string rc_name="Resource";
   uint   rc_width=100;
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- set the size of the pixel array
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- fill the pixel array with a transparent color and create a graphical resource ba
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- check the created graphical resource.
//--- get the time and price data of the current bar
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- create the Bitmap object using the coordinates of the last tick price and time
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- set the width and height of the created bitmap object equal to the width and he
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- specify the previously created graphical resource for the bitmap object as an i
//--- in this case, in order to indicate the name of the graphical resource used, we
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- set the DodgerBlue color with the transparency of 200
```

```
   uint clr=ColorToARGB(clrDodgerBlue,200);
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(rc_data,clr);
//--- update the graphical resource data
   Update(rc_name,rc_data,rc_width,rc_height,true);
  }
```

See also

Resources, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

ResourceFree

The function deletes dynamically created resource (freeing the memory allocated for it).

```
bool  ResourceFree(
   const string  resource_name      // resource name
   );
```

Parameters

resource_name

[in] Resource name should start with "::".

Return Value

True if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

ResourceFree() allows mql5 application developers to manage memory consumption when actively working with resources. Graphical objects bound to the resource being deleted from the memory will be displayed correctly after its deletion. However, newly created graphical objects (OBJ_BITMAP and OBJ_BITMAP_LABEL) will not be able to use the deleted resource.

The function deletes only dynamic resources created by the program.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the parameters of the graphical resource
   string rc_name="Resource";
   uint   rc_width=100;
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- set the size of the pixel array
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- fill the pixel array with a transparent color and create a graphical resource ba
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- check the created graphical resource.
//--- get the time and price data of the current bar
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- create the Bitmap object using the coordinates of the last tick price and time
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- set the width and height of the created bitmap object equal to the width and hei
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- specify the previously created graphical resource for the bitmap object as an ir
//--- in this case, in order to indicate the name of the graphical resource used, we r
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- set the DodgerBlue color with the transparency of 200
   uint clr=ColorToARGB(clrDodgerBlue,200);
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(rc_data,clr);
//--- update the graphical resource data
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- wait three seconds and change the image color
   Print("Wait 3 seconds before changing color");
   Sleep(3000);
//--- set the OrangeRed color with the transparency of 200
   Print("Change color");
   clr=ColorToARGB(clrOrangeRed,200);
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(rc_data,clr);
//--- update the graphical resource data
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- wait three seconds and release the graphical resource
```

```
   Print("Wait 3 seconds before ResourceFree()");
   Sleep(3000);
   bool res=ResourceFree("::"+rc_name);
   Print("ResourceFree: ",res);

//--- try changing the color after releasing the resource
   Print("Trying to change color to GreenYellow after ResourceFree()");
//--- set the GreenYellow color with the transparency of 200
   clr=ColorToARGB(clrGreenYellow,200);
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(rc_data,clr);
//--- update the graphical resource data (the image remains, but the color cannot be
   Update(rc_name,rc_data,rc_width,rc_height,true);
   Print("The color has not changed because the resource has been released");

//--- wait three seconds and delete the bitmap object
   Print("Wait 3 seconds before deleting the Bitmap object");
   Sleep(3000);
   Print("Delete Bitmap object");
   ObjectDelete(0,obj_name);
  }
//+------------------------------------------------------------------+
//| Update graphical resource data                                   |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- leave if zero dimensions are passed
   if(width==0 || height==0)
      return;
//--- update resource data and redraw the chart
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
```

See also

[Resources](), [ObjectCreate()](), [PlaySound()](), [ObjectSetString()](), [OBJPROP_BMPFILE]()

ResourceReadImage

The function reads data from the graphical resource created by ResourceCreate() function or saved in EX5 file during compilation.

```
bool  ResourceReadImage(
   const string      resource_name,       // graphical resource name for reading
   uint&             data[],              // array for receiving data from the resourc
   uint&             width,               // for receiving the image width in the reso
   uint&             height,              // for receiving the image height in the res
   );
```

Parameters

resource_name

[in] Name of the graphical resource containing an image. To gain access to its own resources, the name is used in brief form "::resourcename". If we download a resource from a compiled EX5 file, the full name should be used with the path relative to MQL5 directory, file and resource names – "path\\filename.ex5::resourcename".

data[][]

[in] One- or two-dimensional array for receiving data from the graphical resource.

img_width

[out] Graphical resource image width in pixels .

img_height

[out] Graphical resource image height in pixels .

Return Value

true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

If *data[]* array is then to be used for creating a graphical resource, COLOR_FORMAT_ARGB_NORMALIZE or COLOR_FORMAT_XRGB_NOALPHA color formats should be used.

If *data[]* array is two-dimensional and its second dimension is less than X(width) graphical resource size, ResourceReadImage() function returns false and reading is not performed. But if the resource exists, actual image size is returned to width and height parameters. This will allow making another attempt to receive data from the resource.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- declare the parameters of the graphical resource
   string rc_name="Resource";
   uint   rc_width=100;
   uint   rc_height=100;
   uint   rc_data[];
   uint   rc_size=rc_width*rc_height;

   ResetLastError();
//--- set the size of the pixel array
   if(ArrayResize(rc_data,rc_size)!=rc_size)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- fill the pixel array with a transparent color and create a graphical resource ba
   ArrayInitialize(rc_data,0x00FFFFFF);
   if(!ResourceCreate(rc_name,rc_data,rc_width,rc_height,0,0,0,COLOR_FORMAT_ARGB_NORMA
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",rc_data.Size());

//--- check the created graphical resource.
//--- get the time and price data of the current bar
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- create the Bitmap object using the coordinates of the last tick price and time
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- set the width and height of the created bitmap object equal to the width and he
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- specify the previously created graphical resource for the bitmap object as an i
//--- In this case, in order to indicate the name of the graphical resource used, we
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+rc_name);

//--- set the DodgerBlue color with the transparency of 200
   uint clr=ColorToARGB(clrDodgerBlue,200);
```

```
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(rc_data,clr);
//--- update the graphical resource data
   Update(rc_name,rc_data,rc_width,rc_height,true);

//--- wait 3 seconds before reading the graphical resource data
   Print("Wait 3 seconds before ResourceReadImage()");
   Sleep(3000);
//--- read the image from the resource into a new array of pixels
   uint rc_data_copy[];
   uint w=0,h=0;
   ResetLastError();
   if(!ResourceReadImage("::"+rc_name,rc_data_copy,w,h))
     {
      Print("ResourceReadImage() failed. Error code: ",GetLastError());
      return;
     }

//--- set the OrangeRed color with the transparency of 200
   clr=ColorToARGB(clrOrangeRed,200);
//--- fill the entire array of pixels of the graphical resource with the set color an
   ArrayInitialize(rc_data_copy,clr);
   if(!ResourceCreate(rc_name+"Copy",rc_data_copy,rc_width,rc_height,0,0,0,COLOR_FORM
     {
      Print("New ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created new recource array: ",rc_data_copy.Size());

//--- create the "Graphical label" object using the coordinates of the last tick pric
   string obj_name2="BitmapLabel";
   if(!ObjectCreate(0,obj_name2,OBJ_BITMAP_LABEL,0,0,0))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- get screen coordinates using the previously received price and time
   int x=0,y=0;
   if(!ChartTimePriceToXY(0,0,tick.time,tick.bid,x,y))
     {
      Print("New ChartTimePriceToXY() failed. Error code: ",GetLastError());
      return;
     }
//--- set the width and height of the created graphical label object equal to the widt
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name2,OBJPROP_XSIZE,rc_width);
   ObjectSetInteger(0,obj_name2,OBJPROP_YSIZE,rc_height);
   ObjectSetInteger(0,obj_name2,OBJPROP_ANCHOR,ANCHOR_LEFT_UPPER);
   ObjectSetInteger(0,obj_name2,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(0,obj_name2,OBJPROP_YDISTANCE,y);
//--- set the copied graphical resource as an image file for the graphical label objec
//--- in this case, in order to indicate the name of the graphical resource used, we
   ObjectSetString(0,obj_name2,OBJPROP_BMPFILE,"::"+rc_name+"Copy");

//--- change the color of the new graphical label object
   Print("Wait 3 seconds before changing color to GreenYellow");
   Sleep(3000);
//--- set the GreenYellow color with the transparency of 200
   clr=ColorToARGB(clrGreenYellow,200);
//--- fill the entire array of pixels of the new graphical resource with the set colo
   ArrayInitialize(rc_data_copy,clr);
//--- update the graphical resource data
   Update(rc_name+"Copy",rc_data_copy,rc_width,rc_height,true);

//--- wait three seconds and delete resources and both objects
   Print("Wait 3 seconds before deleting both objects");
   Sleep(3000);
   Print("Deleting Resource and all Bitmap objects");
   ResourceFree("::"+rc_name);
   ResourceFree("::"+rc_name+"Copy");
   ObjectDelete(0,obj_name);
   ObjectDelete(0,obj_name2);
  }
//+------------------------------------------------------------------+
//| Update graphical resource data                                   |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- leave if zero dimensions are passed
   if(width==0 || height==0)
      return;
//--- update resource data and redraw the chart
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
```

See also

[Resource](), [ObjectCreate()](), [ObjectSetString()](), [OBJPROP_BMPFILE]()

ResourceSave

Saves a resource into the specified file.

```
bool  ResourceSave(
   const string  resource_name      // Resource name
   const string  file_name          // File name
   );
```

Parameters

resource_name

[in] The name of the resource, must start with "::".

file_name

[in] The name of the file relative to MQL5\Files.

Return Value

true - in case of success, otherwise false. For the error information call GetLastError().

Note

The function always overwrites a file and creates all the required intermediate directories in the file name if necessary.

Example:

```
//--- graphical resource parameters
string ExtResName="Resource";
int    ExtResWidth=100;
int    ExtResHeight=100;
uint   ExtResData[];
int    ExtResSize=ExtResWidth*ExtResHeight;

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   ResetLastError();
//--- set the size of the pixel array
   if(ArrayResize(ExtResData,ExtResSize)!=ExtResSize)
     {
      Print("ArrayResize() failed. Error code: ",GetLastError());
      return;
     }
//--- fill the pixel array with a transparent color and create a graphical resource ba
   ArrayInitialize(ExtResData,0x00FFFFFF);
   if(!ResourceCreate(ExtResName,ExtResData,ExtResWidth,ExtResHeight,0,0,0,COLOR_FORM
     {
      Print("ResourceCreate() failed. Error code: ",GetLastError());
      return;
     }
   Print("Size of created recource array: ",ExtResData.Size());

//--- check the created graphical resource.
//--- get the time and price data of the current bar
   MqlTick tick={};
   if(!SymbolInfoTick(Symbol(),tick))
     {
      Print("SymbolInfoTick() failed. Error code: ",GetLastError());
      return;
     }
//--- Create the Bitmap object using the coordinates of the last tick price and time
   string obj_name="Bitmap";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP,0,tick.time,tick.bid))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- set the width and height of the created bitmap object equal to the width and he
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name,OBJPROP_XSIZE,ExtResWidth);
   ObjectSetInteger(0,obj_name,OBJPROP_YSIZE,ExtResHeight);
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- specify the previously created graphical resource for the bitmap object as an i
//--- in this case, in order to indicate the name of the graphical resource used, we
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"::"+ExtResName);

//--- set the GreenYellow color with the transparency of 200
   uint clr=ColorToARGB(clrGreenYellow,200);
//--- fill the entire array of pixels of the graphical resource with the set color
   ArrayInitialize(ExtResData,clr);
//--- draw the grid using DodgerBlue color
   for(int x=0;x<ExtResWidth;x+=9)
      LineVertical(x,0,ExtResHeight,ColorToARGB(clrDodgerBlue,200));
   for(int y=0;y<ExtResHeight;y+=9)
      LineHorizontal(0,ExtResWidth,y,ColorToARGB(clrDodgerBlue,200));
//--- update the graphical resource data
   Update(ExtResName,ExtResData,ExtResWidth,ExtResHeight,true);

//--- wait 3 seconds before deleting the resource and graphical object
   Print("Wait 3 seconds before deleting the resource and graphic object");
   Sleep(3000);

//--- save the resource to the file
   ResetLastError();
   if(!ResourceSave("::"+ExtResName,"ResourceSave\\"+ExtResName+".bmp"))
```

```
     {
      Print("ResourceSave() failed. Error code: ",GetLastError());
      return;
     }
//--- delete the resource and graphic object image
   if(!ResourceFree("::"+ExtResName))
     {
      Print("ResourceFree() failed. Error code: ",GetLastError());
      return;
     }
   if(!ObjectDelete(0,obj_name))
     {
      Print("ObjectDelete() failed. Error code: ",GetLastError());
      return;
     }

//--- now create the "Graphical label" object using the coordinates of the price and 
//--- get screen coordinates using the previously received last tick price and time
   int x=0,y=0;
   if(!ChartTimePriceToXY(0,0,tick.time,tick.bid,x,y))
     {
      Print("ChartTimePriceToXY() failed. Error code: ",GetLastError());
      return;
     }
   obj_name="BitmapLabel";
   if(!ObjectCreate(0,obj_name,OBJ_BITMAP_LABEL,0,0,0))
     {
      Print("ObjectCreate() failed. Error code: ",GetLastError());
      return;
     }
//--- set the object anchor point to its center.
   ObjectSetInteger(0,obj_name,OBJPROP_ANCHOR,ANCHOR_CENTER);
//--- set the coordinates of the price and time of the last tick, shifted to the left
   ObjectSetInteger(0,obj_name,OBJPROP_XDISTANCE,x-ExtResWidth);
   ObjectSetInteger(0,obj_name,OBJPROP_YDISTANCE,y);
//--- specify the resource, previously saved as a bmp image, for the graphical label 
   ObjectSetString(0,obj_name,OBJPROP_BMPFILE,"\\Files\\ResourceSave\\"+ExtResName+".
//--- update the chart
   ChartRedraw();

//--- wait 3 seconds before deleting the graphical label object
   Print("Wait 3 seconds before deleting the new graphic object");
   Sleep(3000);
   if(!ObjectDelete(0,obj_name))
      Print("ObjectDelete() failed. Error code: ",GetLastError());
  }
//+------------------------------------------------------------------+
//| Update graphical resource data                                   |
//+------------------------------------------------------------------+
void Update(const string res_name,const uint &pixel_data[],const uint width,const uint
  {
//--- leave if zero dimensions are passed
   if(width==0 || height==0)
      return;
//--- update resource data and redraw the chart
   if(ResourceCreate(res_name,pixel_data,width,height,0,0,0,COLOR_FORMAT_ARGB_NORMALI
      ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Draw a vertical line                                             |
//+------------------------------------------------------------------+
void LineVertical(int x,int y1,int y2,const uint clr)
  {
   int tmp;
//--- sort by Y
   if(y1>y2)
     {
      tmp=y1;
      y1 =y2;
      y2 =tmp;
     }
//--- line outside the image boundaries
   if(y2<0 || y1>=ExtResHeight || x<0 || x>=ExtResWidth)
      return;
//--- stay within the image boundaries
   if(y1<0)
      y1=0;
   if(y2>=ExtResHeight)
      y2=ExtResHeight-1;
//--- draw the line
   int index=y1*ExtResWidth+x;
   for(int i=y1; i<=y2; i++,index+=ExtResWidth)
      ExtResData[index]=clr;
  }
//+------------------------------------------------------------------+
//| Draw a horizontal line                                           |
//+------------------------------------------------------------------+
void LineHorizontal(int x1,int x2,int y,const uint clr)
  {
   int tmp;
//--- sort by X
   if(x1>x2)
     {
      tmp=x1;
      x1 =x2;
      x2 =tmp;
     }
//--- line outside the image boundaries
```

```
   if(x2<0 || x1>=ExtResWidth || y<0 || y>=ExtResHeight)
      return;
//--- stay within the image boundaries
   if(x1<0)
      x1=0;
   if(x2>=ExtResWidth)
      x2=ExtResWidth-1;
//--- draw the line
   ArrayFill(ExtResData,y*ExtResWidth+x1,(x2-x1)+1,clr);
  }
```

See also

Resources, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

SetReturnError

Sets the code that returns the terminal process when completing the operation.

```
void  SetReturnError(
   int ret_code       // client terminal completion code
   );
```

Parameters

ret_code

[in] The code to be returned by the client terminal process when completing the operation.

Return Value

No return value.

Note

Setting the specified *ret_code* return code using the SetReturnError() function is useful for analyzing reasons of the programmatic operation completion when launching the terminal via the command line.

Unlike TerminalClose(), the SetReturnError() function does not complete the terminal operation. Instead, it only sets the code that returns the terminal process upon its completion.

If the SetReturnError() function is called multiple times and/or from different MQL5 programs, the terminal returns the last set return code.

The set code is returned upon the terminal process completion except for the following cases:

- a critical error has occurred during execution;
- the TerminalClose(int ret_code) function issuing the terminal operation completion command with a specified code has been called.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   matrix matrix_a =
     {
        {-3.474589, 1.106384, -9.091977,-3.925227 },
        {-5.522139, 2.366887,-15.162351,-6.357512 },
        { 8.394926,-2.960067, 22.292115, 9.524129 },
        { 7.803242,-2.080287, 19.217706, 8.186645 }
     };
   matrix matrix_l(4,4);
   matrix matrix_u(4,4);

//--- LU decomposition
```

```
   matrix_a.LU(matrix_l,matrix_u);

//--- check if A = L * U
   matrix matrix_lu=matrix_l.MatMul(matrix_u);
   int    compare_errors=(int)matrix_a.Compare(matrix_lu,1e-29);
   Print("MatrixCompare errors=",compare_errors);

//--- upon completion, the client terminal will return the number of errors in compar
   SetReturnError(compare_errors);
  }
```

See also

Program Running, Runtime Errors, Uninitialization Reason Codes, TerminalClose

SetUserError

Sets the predefined variable _LastError into the value equal to ERR_USER_ERROR_FIRST + user_error

```
void  SetUserError(
   ushort user_error,    // error number
   );
```

Parameters

user_error

[in] Error number set by a user.

Return Value

No return value.

Note

After an error has been set using the SetUserError(user_error) function, GetLastError() returns value equal to ERR_USER_ERROR_FIRST + user_error.

Example:

```
void OnStart()
  {
//--- set error number 65537=(ERR_USER_ERROR_FIRST +1)
   SetUserError(1);
//--- get last error code
   Print("GetLastError = ",GetLastError());
/*
   Result
   GetLastError = 65537
*/
  }
```

Sleep

The function suspends execution of the current Expert Advisor or script within a specified interval.

```
void  Sleep(
   int  milliseconds      // interval
   );
```

Parameters

milliseconds

[in] Delay interval in milliseconds.

Return Value

No return value.

Note

The Sleep() function can't be called for custom indicators, because indicators are executed in the interface thread and must not slow down it. The function has the built-in check of EA halt flag every 0.1 seconds.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- display a countdown from 10 to 1 in the comments on the chart
   for(int i=10; i>0 && !IsStopped(); i--)
     {
      Comment(StringFormat("Wait %u seconds",i));
      Sleep(1000);
     }
//--- write a text in the "arriving" comment that describes the purpose of the script
   string text="This was a test showing how the Sleep() function works";
   string mess="";
   for(int i=0; i<(int)text.Length(); i++)
     {
      mess+=ShortToString(text.GetChar(i));
      Sleep(100);
      Comment(mess);
     }
//--- say goodbye...
   Sleep(1000);
   for(int i=0; i<6; i++)
     {
      mess=(i % 2 == 0 ? "" : "  Bye!");
      Comment(mess);
      Sleep(300);
     }
//--- delete the text on the chart
   Comment("");
  }
```

TerminalClose

The function commands the terminal to complete operation.

```
bool  TerminalClose(
   int ret_code      // closing code of the client terminal
   );
```

Parameters

ret_code

[in] Return code, returned by the process of the client terminal at the operation completion.

Return Value

The function returns true on success, otherwise - false.

Note

The TerminalClose() function does not stop the terminal immediately, it just commands the terminal to complete its operation.

The code of an Expert Advisor that called TerminalClose() must have all arrangements for the immediate completion (e.g. all previously opened files must be closed in the normal mode). Call of this function must be followed by the return operator.

The *ret_code* parameter allows indicating the necessary return code for analyzing reasons of the program termination of the terminal operation when starting it from the command prompt.

Example:

```
//--- input parameters
input int   tiks_before=500; // number of ticks till termination
input int   pips_to_go=15;   // distance in pips
input int   seconds_st=50;   // number of seconds given to the Expert Advisor
//--- globals
datetime    launch_time;
int         tick_counter=0;
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(__FUNCTION__," reason code = ",reason);
   Comment("");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   static double first_bid=0.0;
   MqlTick        tick;
   double         distance;
//---
   SymbolInfoTick(_Symbol,tick);
   tick_counter++;
   if(first_bid==0.0)
     {
      launch_time=tick.time;
      first_bid=tick.bid;
      Print("first_bid =",first_bid);
      return;
     }
//--- price distance in pips
   distance=(tick.bid-first_bid)/_Point;
//--- show a notification to track the EA operation
   string comm="From the moment of start:\r\n\x25CF elapsed seconds: "+
               IntegerToString(tick.time-launch_time)+" ;"+
               "\r\n\x25CF ticks received: "+(string)tick_counter+" ;"+
               "\r\n\x25CF price went in points: "+StringFormat("%G",distance);
   Comment(comm);
//--- section for checking condition to close the terminal
   if(tick_counter>=tiks_before)
      TerminalClose(0);    // exit by tick counter
   if(distance>pips_to_go)
      TerminalClose(1);    // go up by the number of pips equal to pips_to_go
   if(distance<-pips_to_go)
      TerminalClose(-1);   // go down by the number of pips equal to pips_to_go
   if(tick.time-launch_time>seconds_st)
      TerminalClose(100);  // termination by timeout
//---
  }
```

See also

Program running, Execution errors, Reasons for deinitialization

TesterHideIndicators

Sets the mode of displaying/hiding indicators used in an EA. The function is intended for managing the visibility of used indicators only during testing.

```
void  TesterHideIndicators(
   bool       hide      // flag
   );
```

Parameters

hide

[in] Flag for hiding indicators when testing. Set true to hide created indicators, otherwise false.

Return Value

None.

Note

By default, all indicators created in a tested EA are displayed on the visual testing chart. Besides, these indicators are displayed on the chart that is automatically opened when testing is complete. The TesterHideIndicators() function allows developers to implement the ability to disable the display of used indicators.

To disable the display of an applied indicator when testing an EA, call TesterHideIndicators() equal to true before creating the EA's handle – all indicators created after that are marked with the hide flag. These indicators are not displayed during a visual test and on the chart that is automatically opened upon completion of the test.

To disable the hide mode of the newly created indicators, call TesterHideIndicators() equal to false. Only indicators generated directly from the tested EA can be displayed on the testing chart. This rule applies only to cases where there is not a single template in <data_folder>MQL5\Profiles\Templates.

If the <data_folder>MQL5\Profiles\Templates directory contains a special template <EA_name>.tpl, only the indicators from this template are displayed during a visual testing and on the testing chart. In this case, no indicators applied in the tested EA are displayed. This behavior remains even if TesterHideIndicators() equal to true is called in the EA code.

If the <data_folder>MQL5\Profiles\Templates directory contains no special <EA_name>.tpl template having tester.tpl instead, indicators from the tester.tpl and the ones from the EA not disabled by the TesterHideIndicators() function are displayed during a visual testing and on the testing chart. If there is no tester.tpl template, indicators from the default.tpl template are used instead.

If the strategy tester finds no suitable template (<EA_name>.tpl, tester.tpl or default.tpl), display of the indicators applied in the EA is fully managed by the TesterHideIndicators() function.

Example:

```
bool CSampleExpert::InitIndicators(void)
  {
   TesterHideIndicators(true);
//--- create MACD indicator
   if(m_handle_macd==INVALID_HANDLE)
      if((m_handle_macd=iMACD(NULL,0,12,26,9,PRICE_CLOSE))==INVALID_HANDLE)
```

```
        {
         printf("Error creating MACD indicator");
         return(false);
        }
   TesterHideIndicators(false);
//--- create EMA indicator and add it to collection
   if(m_handle_ema==INVALID_HANDLE)
      if((m_handle_ema=iMA(NULL,0,InpMATrendPeriod,0,MODE_EMA,PRICE_CLOSE))==INVALID_
        {
         printf("Error creating EMA indicator");
         return(false);
        }
//--- succeed
   return(true);
  }
```

See also

IndicatorRelease

TesterStatistics

The function returns the value of the specified statistical parameter calculated based on testing results.

```
double  TesterStatistics(
   ENUM_STATISTICS statistic_id      // ID
   );
```

Parameters

statistic_id

[in] The ID of the statistical parameter from the ENUM_STATISTICS enumeration.

Return Value

The value of the statistical parameter from testing results.

Note

The function can be called inside OnTester() or OnDeinit() in the tester. In other cases the result is undefined.

Example:

```
// The EA based on standard "MACD Sample.mq5" file
// shows the result of TesterStatistics() in the Tester event handler
#define MACD_MAGIC 1234502
//---
#include <Trade\Trade.mqh>
#include <Trade\SymbolInfo.mqh>
#include <Trade\PositionInfo.mqh>
#include <Trade\AccountInfo.mqh>
//---
input double InpLots          =0.1; // Lots
input int    InpTakeProfit    =50;  // Take Profit (in pips)
input int    InpTrailingStop  =30;  // Trailing Stop Level (in pips)
input int    InpMACDOpenLevel =3;   // MACD open level (in pips)
input int    InpMACDCloseLevel=2;   // MACD close level (in pips)
input int    InpMATrendPeriod =26;  // MA trend period
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit(void)
  {
//--- create all the necessary objects
   if(!ExtExpert.Init())
      return(INIT_FAILED);
//--- successful initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert new tick handling function                                |
//+------------------------------------------------------------------+
void OnTick(void)
  {
   static datetime limit_time=0; // last call time considering 'timeout'
//--- if the time exceeds the specified limit_time value
   if(TimeCurrent()>=limit_time)
     {
      //--- check the data
      if(Bars(Symbol(),Period())>2*InpMATrendPeriod)
        {
         //--- if successful, increase limit_time by 'timeout' seconds
         if(ExtExpert.Processing())
            limit_time=TimeCurrent()+ExtTimeOut;
        }
     }
  }
//+------------------------------------------------------------------+
//| Expert tester handling function                                  |
//+------------------------------------------------------------------+
double OnTester(void)
  {
   double ret=TesterStatistics(STAT_PROFIT_FACTOR);
   double profit=TesterStatistics(STAT_PROFIT);
   int    trades_total=(int)TesterStatistics(STAT_TRADES);
   int    profit_total=(int)TesterStatistics(STAT_PROFIT_TRADES);
   int    loss_total=(int)TesterStatistics(STAT_LOSS_TRADES);
   PrintFormat("%s: Profit = %.2f, trades total: %lu, profit trades total: %lu, loss
   return(ret);
   /*
   Result:
   OnTester: Profit = 209.84, trades total: 13, profit trades total: 11, loss trades
   final balance 10209.84 USD
   OnTester result 3.020606644198363
   */
  }
```

TesterStop

Gives program operation completion command when testing.

```
void  TesterStop();
```

Return Value

No return value.

Note

The TesterStop() function is designed for a routine early shutdown of an EA on a test agent – for example, when reaching a specified number of losing trades or a preset drawdown level.

TesterStop() call is considered a normal completion of a test, therefore the OnTester() function is called, and the entire accumulated trading statistics and optimization criterion value are submitted to the strategy tester.

Calling ExpertRemove() in the strategy tester also means normal test completion and allows for obtaining trading statistics, but the EA is unloaded from the agent’s memory. In this case, performing a pass on the next set of parameters requires time in order to reload the program. Therefore, TesterStop() is a preferred option for an early routine completion of a test.

Example:

```
//--- defines
#define BALANCE_LOSS_STOP  100.0       // value of the balance drawdown, at which test
#define EQUITY_LOSS_STOP   100.0       // value of the equity drawdown, at which testi

//--- input parameters
input  double  InpLots        =  0.1;  // lots
input  uint    InpStopLoss    =  50;   // Stop loss in points
input  uint    InpTakeProfit  =  150;  // Take Profit in points
sinput ulong   InpMagic       =  123;  // MagicNumber
sinput ulong   InpDeviation   =  5;    // deviation
//--- global variables
CTrade      trade;                     // trade class instance
CSymbolInfo symb;                      // symbol class instance
CAccountInfo account;                  // trading account class instance
...
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- successful initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- update current quotes
   if(!symb.RefreshRates())
      return;
   ...

//--- if the balance or equity have dropped more than indicated in the BALANCE_LOSS_ST
//--- the test is considered unsuccessful and the TesterStop() function is called
//--- check for loss of balance by more than BALANCE_LOSS_STOP
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()<balance_prev-BALANCE_LOSS_STOP)
        {
         PrintFormat("The initial balance of %.2f %s decreased by %.2f %s, and now ha
         TesterStop();
         /*
         Result:
         The initial balance of 10000.00 USD decreased by 100.10 USD, and now has a va
         TesterStop() called on 9% of testing interval
         */
        }
     }
//--- check the loss of equity by more than EQUITY_LOSS_STOP
   if(equity_prev!=account.Equity())
     {
      if(account.Equity()<equity_prev-EQUITY_LOSS_STOP)
        {
         PrintFormat("The initial equity of %.2f %s decreased by %.2f %s, and now has
         TesterStop();
         /*
         Result:
         The initial equity of 10000.00 USD decreased by 100.10 USD, and now has a val
         TesterStop() called on 9% of testing interval
         */
        }
     }
  }
```

See also

Program Running, Testing Trading Strategies, ExpertRemove, SetReturnError

TesterDeposit

The special function that emulates depositing funds during a test. It can be used in some money management systems.

```
bool  TesterDeposit(
   double money      // deposited sum
   );
```

Parameters

money

[in] Money to be deposited to an account in the deposit currency.

Return Value

Returns true if successful, otherwise - false.

Example:

```
//--- defines
#define BALANCE_LOSS_DEPOSIT  100.0    // value of the balance drawdown, at which func

//--- input parameters
input  double  InpLots        =  0.1;  // Lots
input  uint    InpStopLoss    =  50;   // Stop loss in points
input  uint    InpTakeProfit  =  150;  // Take Profit in points
sinput ulong   InpMagic       =  123;  // Magic number
sinput ulong   InpDeviation   =  5;    // Deviation
//--- global variables
CTrade      trade;                     // trade class instance
CSymbolInfo symb;                      // symbol class instance
CAccountInfo account;                  // trading account class instance
...
double      balance_dep_summ;          // total amount of balance top-ups
uint        balance_dep_total;         // number of balance top-ups
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- save the initial balance values
   balance_prev=account.Balance();
   balance_dep_summ=0;
   balance_dep_total=0;
//--- successful initialization
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- update current quotes
   if(!symb.RefreshRates())
      return;
   ...

//--- if the balance has dropped more than indicated in the BALANCE_LOSS_DEPOSIT macro
//--- it is necessary to top up the account and call the TesterDeposit() function
//--- check for loss of balance by more than BALANCE_LOSS_DEPOSIT
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()<balance_prev-BALANCE_LOSS_DEPOSIT)
        {
         double loss=balance_prev-account.Balance();
         PrintFormat("The initial balance of %.2f %s decreased by %.2f %s. It is neces
         if(TesterDeposit(loss))
           {
            balance_dep_total++;
            balance_dep_summ+=loss;
            balance_prev=account.Balance();
            PrintFormat("Funds have been deposited into the account. Account balance:
            PrintFormat("Total deposits: %lu. Amount of deposits: %.2f %s.",balance_de
           }
         /*
         Result:
         The initial balance of 10000.00 USD decreased by 116.00 USD. It is necessary
         deal #45 balance 116.00 [deposit] done
         Funds have been deposited into the account. Account balance: 10000.00 USD.
         Total deposits: 1. Amount of deposits: 116.00 USD.
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| Tester function                                                  |
//+------------------------------------------------------------------+
double OnTester()
  {
//--- set the maximum balance drawdown in monetary terms as the output handler value
   double ret=TesterStatistics(STAT_BALANCE_DD);
//--- display a message about the drawdown, the number of deposits and their total amo
   PrintFormat("%s: Maximum balance drawdown in money: %.2f %s. Total deposits: %lu. A
//--- return the result
   return(ret);
   /*
```

```
    Result:
    OnTester: Maximum balance drawdown in money: 5188.50 USD. Total deposits: 46. Amour
    final balance 4867.50 USD
    OnTester result 5188.5
    */
  }
```

See also

TesterWithdrawal

TesterWithdrawal

The special function to emulate the operation of money withdrawal in the process of testing. Can be used in some asset management systems.

```
bool  TesterWithdrawal(
   double money      // the sum to withdraw
   );
```

Parameters

money

[in] The sum of money that we need to withdraw (in the deposit currency).

Return Value

If successful, returns true, otherwise - false.

Example:

```
//--- defines
#define BALANCE_PROFIT_WITHDRAWAL   5  // the value of the balance profit, at which fu

//--- input parameters
input  double  InpLots        =  0.1;  // Lots
input  uint    InpStopLoss    =  50;   // Stop loss in points
input  uint    InpTakeProfit  =  150;  // Take Profit in points
sinput ulong   InpMagic       =  123;  // Magic number
sinput ulong   InpDeviation   =  5;    // Deviation
//--- global variables
CTrade      trade;                     // trade class instance
CSymbolInfo symb;                      // symbol class instance
CAccountInfo account;                  // trading account class instance
...
double      balance_op_sum;            // total amount of balance operations
uint        balance_op_total;          // number of balance operations
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
   ...
//--- save the initial balance values
   balance_prev=account.Balance();
   balance_op_sum=0;
   balance_op_total=0;
//--- successful initialization
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- update current quotes
   if(!symb.RefreshRates())
      return;
   ...

//--- if the balance profit exceeds the current balance by the value specified in the
//--- it is necessary to withdraw these funds from the account. Call the TesterWithdra
//--- check the balance profit for exceeding BALANCE_PROFIT_WITHDRAWAL
   if(balance_prev!=account.Balance())
     {
      if(account.Balance()>balance_prev+BALANCE_PROFIT_WITHDRAWAL)
        {
         double profit=account.Balance()-balance_prev;
         PrintFormat("The account balance has been increased by %.2f %s. Need to withd
         if(TesterWithdrawal(profit))
           {
            balance_op_total++;
            balance_op_summ+=profit;
            balance_prev=account.Balance();
            PrintFormat("Funds have been withdrawn from the account. Account balance:
            PrintFormat("Total withdrawals: %lu. Amount of withdrawals: %.2f %s.",bala
           }
         /*
         Result:
         The account balance has been increased by 21.00 USD. Need to withdraw these f
         deal #13 balance -21.00 [withdrawal] done
         Funds have been withdrawn from the account. Account balance: 10000.00 USD.
         Total withdrawals: 1. Amount of withdrawals: 21.00 USD.
         */
        }
     }
  }
//+------------------------------------------------------------------+
//| Tester function                                                  |
//+------------------------------------------------------------------+
double OnTester()
  {
//--- set the maximum balance drawdown in monetary terms as the output handler value
   double ret=TesterStatistics(STAT_BALANCE_DD);
//--- display a message about the drawdown, the number of withdrawals and their total
   PrintFormat("%s: Maximum balance drawdown in money: %.2f %s. Total withdrawals: %lu
//--- return the result
   return(ret);
   /*
```

```
    Result:
    OnTester: Maximum balance drawdown in money: 5188.50 USD. Total withdrawals: 2. Amo
    final balance 4867.50 USD
    OnTester result 5188.5
    */
  }
```

See also

TesterDeposit

TranslateKey

Returns a Unicode character by a virtual key code considering the current input language and the status of control keys.

```
short  TranslateKey(
   int  key_code      // key code for receiving a Unicode character
   );
```

Parameters

key_code

[in] Key code.

Return Value

Unicode character in case of a successful conversion. The function returns -1 in case of an error.

Note

The function uses ToUnicodeEx to convert keys pressed by a user into Unicode characters. An error may occur in case ToUnicodeEx is not triggered – for example, when trying to receive the SHIFT key character.

Example:

```
void OnChartEvent(const int id,const long& lparam,const double& dparam,const string& s
  {
   if(id==CHARTEVENT_KEYDOWN)
     {
      short sym=TranslateKey((int)lparam);
      //--- if the entered character is successfully converted to Unicode
      if(sym>0)
         Print(sym,"'",ShortToString(sym),"'");
      else
         Print("Error in TranslateKey for key=",lparam);
     }
  }
```

See also

Client terminal events, OnChartEvent

ZeroMemory

The function resets a variable passed to it by reference.

```
void  ZeroMemory(
   void & variable      // reset variable
   );
```

Parameters

variable

[in] [out] Variable passed by reference, you want to reset (initialize by zero values).

Return Value

No return value.

Note

If the function parameter is a string, the call will be equivalent to indicating NULL as its value.
For simple types and their arrays, as well as for structures/classes consisting of such types, this is a simple reset.
For objects containing strings and dynamic arrays, ZeroMemory() is called for each element.
For any arrays not protected by the const modifier, this is the zeroing of all elements.
For arrays of complex objects, ZeroMemory() is called for each element.

ZeroMemory() can't be applied to classes with protected members or inheritance.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the string
   string str="Test ZeroMemory func";
//--- send the line to the log before applying ZeroMemory() to it
   PrintFormat("The line before applying ZeroMemory() to it: '%s'",str);
//--- reset the string and send the result to the log
   ZeroMemory(str);
   Print("The same line after applying ZeroMemory() to it: '",str,"'");
  /*
   Result:
   The line before applying ZeroMemory() to it: 'Test ZeroMemory func'
   The same line after applying ZeroMemory() to it: ''
  */

//--- declare and initialize the variable of int type
   int var=123;
//--- send the line to the log before applying ZeroMemory() to it
```

```
   PrintFormat("\nThe integer variable before applying ZeroMemory() to it: %d",var);
//--- reset the variable and send the result to the log
   ZeroMemory(var);
   PrintFormat("The same variable after applying ZeroMemory() to it: %d",var);
  /*
   Result:
   The integer variable before applying ZeroMemory() to it: 123
   The same variable after applying ZeroMemory() to it: 0
  */

//--- declare and initialize the array of int type
   int arr[]={0,1,2,3,4,5,6,7,8,9};
//--- send the array to the log before applying ZeroMemory() to it
   Print("\nThe integer array before applying ZeroMemory() to it:");
   ArrayPrint(arr);
//--- reset the array and send the result to the log
   ZeroMemory(arr);
   Print("The same array after applying ZeroMemory() to it:");
   ArrayPrint(arr);
  /*
   Result:
   The integer array before applying ZeroMemory() to it:
   0 1 2 3 4 5 6 7 8 9
   The same array after applying ZeroMemory() to it:
   0 0 0 0 0 0 0 0 0 0
  */

//--- declare a structure of two fields - string and integer ones
   struct STest
     {
      string   var_string;
      long     var_long;
     };
//--- declare and initialize the array of STest structure type
   STest arr_struct[]={ {"0",0}, {"1",1}, {"2",2}, {"3",3} };
//--- send the array to the log before applying ZeroMemory() to it
   Print("\nThe array struct before applying ZeroMemory() to it:");
   ArrayPrint(arr_struct);
//--- reset the structure array and send the result to the log
   ZeroMemory(arr_struct);
   Print("The same array struct after applying ZeroMemory() to it:");
   ArrayPrint(arr_struct);
  /*
   Result:
   The array struct before applying ZeroMemory() to it:
       [var_string] [var_long]
   [0] "0"                   0
   [1] "1"                   1
   [2] "2"                   2
```

```
    [3] "3"                   3
    The same array struct after applying ZeroMemory() to it:
        [var_string] [var_long]
    [0] null                  0
    [1] null                  0
    [2] null                  0
    [3] null                  0
   */
   }
```

Group of Functions for Working with Arrays

Arrays are allowed to be maximum four-dimensional. Each dimension is indexed from 0 to *dimension_size-1*. In a particular case of a one-dimensional array of 50 elements, calling of the first element will appear as array[0], of the last one - as array[49].

Function	Action
ArrayBsearch	Returns index of the first found element in the first array dimension
ArrayCopy	Copies one array into another
ArrayCompare	Returns the result of comparing two arrays of simple types or custom structures without complex objects
ArrayFree	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.
ArrayGetAsSeries	Checks direction of array indexing
ArrayInitialize	Sets all elements of a numeric array into a single value
ArrayFill	Fills an array with the specified value
ArrayIsSeries	Checks whether an array is a timeseries
ArrayIsDynamic	Checks whether an array is dynamic
ArrayMaximum	Search for an element with the maximal value
ArrayMinimum	Search for an element with the minimal value
ArrayPrint	Prints an array of a simple type or a simple structure into journal
ArrayRange	Returns the number of elements in the specified dimension of the array
ArrayResize	Sets the new size in the first dimension of the array
ArrayInsert	Inserts the specified number of elements from a source array to a receiving one starting from a specified index
ArrayRemove	Removes the specified number of elements from the array starting with a specified index
ArrayReverse	Reverses the specified number of elements in the array starting with a specified index
ArraySetAsSeries	Sets the direction of array indexing
ArraySize	Returns the number of elements in the array
ArraySort	Sorting of numeric arrays by the first dimension
ArraySwap	Swaps the contents of two dynamic arrays of the same type
ArrayToFP16	Copies an array of type float or double into an array of type ushort with the given format

Function	Action
ArrayToFP8	Copies an array of type float or double into an array of type uchar with the given format
ArrayFromFP16	Copies an array of type ushort into an array of float or double type with the given format
ArrayFromFP8	Copies an array of type uchar into an array of float or double type with the given format

ArrayBsearch

Searches for a specified value in a multidimensional numeric array sorted ascending. Search is performed through the elements of the first dimension.

For searching in an array of double type

```
int  ArrayBsearch(
   const double&     array[],    // array for search
   double            value       // what is searched for
   );
```

For searching in an array of float type

```
int  ArrayBsearch(
   const float&     array[],    // array for search
   float            value       // what is searched for
   );
```

For searching in an array of long type

```
int  ArrayBsearch(
   const long&     array[],    // array for search
   long            value       // what is searched for
   );
```

For searching in an array of int type

```
int  ArrayBsearch(
   const int&     array[],    // array for search
   int            value       // what is searched for
   );
```

For searching in an array of short type

```
int  ArrayBsearch(
   const short&     array[],    // array for search
   short            value       // what is searched for
   );
```

For searching in an array of char type

```
int  ArrayBsearch(
   const char&     array[],    // array for search
   char            value       // what is searched for
   );
```

Parameters

array[]

[in] Numeric array for search.

value

[in] Value for search.

Return Value

The function returns index of a found element. If the wanted value isn't found, the function returns the index of an element nearest in value.

Note

Binary search processes only sorted arrays. To sort numeric arrays use the ArraySort() function.

Example:

```
#property description "Script based on RSI indicator data displays"
#property description "how often the market was in"
#property description "overbought and oversold areas in the specified time interval."
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input int                InpMAPeriod=14;                    // Moving average period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;       // Price type
input double             InpOversoldValue=30.0;             // Oversold level
input double             InpOverboughtValue=70.0;           // Overbought level
input datetime           InpDateStart=D'2012.01.01 00:00';  // Analysis start date
input datetime           InpDateFinish=D'2013.01.01 00:00'; // Analysis finish date
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double rsi_buff[]; // array of the indicator values
   int    size=0;     // array size
//--- receive RSI indicator handle
   ResetLastError();
   int rsi_handle=iRSI(Symbol(),Period(),InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- failed to receive the indicator handle
      PrintFormat("Indicator handle receiving error. Error code = %d",GetLastError());
      return;
     }
//--- being in the loop, until the indicator calculates all its values
   while(BarsCalculated(rsi_handle)==-1)
     {
      //--- exit if the indicator has forcedly completed the script's operation
      if(IsStopped())
         return;
      //--- a pause to allow the indicator to calculate all its values
      Sleep(10);
     }
//--- copy the indicator values for a certain period of time
```

```
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,InpDateFinish,rsi_buff)==-1)
     {
      PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError
      return;
     }
//--- receive the array size
   size=ArraySize(rsi_buff);
//--- sort out the array
   ArraySort(rsi_buff);
//--- find out the time (in percentage terms) the market was in the oversold area
   double ovs=(double)ArrayBsearch(rsi_buff,InpOversoldValue)*100/(double)size;
//--- find out the time (in percentage terms) the market was in the overbought area
   double ovb=(double)(size-ArrayBsearch(rsi_buff,InpOverboughtValue))*100/(double)si
//--- form the strings for displaying the data
   string str="From "+TimeToString(InpDateStart,TIME_DATE)+" to "
              +TimeToString(InpDateFinish,TIME_DATE)+" the market was:";
   string str_ovb="in overbought area "+DoubleToString(ovb,2)+"% of time";
   string str_ovs="in oversold area "+DoubleToString(ovs,2)+"% of time";
//--- display the data on the chart
   CreateLabel("top",5,60,str,clrDodgerBlue);
   CreateLabel("overbought",5,35,str_ovb,clrDodgerBlue);
   CreateLabel("oversold",5,10,str_ovs,clrDodgerBlue);
//--- redraw the chart
   ChartRedraw(0);
//--- pause
   Sleep(10000);
  }
//+------------------------------------------------------------------+
//| Display comment in the bottom left corner of the chart           |
//+------------------------------------------------------------------+
void CreateLabel(const string name,const int x,const int y,
                 const string str,const color clr)
  {
//--- create the label
   ObjectCreate(0,name,OBJ_LABEL,0,0,0);
//--- bind the label to the bottom left corner
   ObjectSetInteger(0,name,OBJPROP_CORNER,CORNER_LEFT_LOWER);
//--- change position of the anchor point
   ObjectSetInteger(0,name,OBJPROP_ANCHOR,ANCHOR_LEFT_LOWER);
//--- distance from the anchor point in X-direction
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
//--- distance from the anchor point in Y-direction
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
//--- label text
   ObjectSetString(0,name,OBJPROP_TEXT,str);
//--- text color
   ObjectSetInteger(0,name,OBJPROP_COLOR,clr);
//--- text size
```

```
   ObjectSetInteger(0,name,OBJPROP_FONTSIZE,12);
  }
```

ArrayCopy

It copies an array into another one.

```
int  ArrayCopy(
   void&         dst_array[],         // destination array
   const void&   src_array[],         // source array
   int           dst_start=0,         // index starting from which write into destinat
   int           src_start=0,         // first index of a source array
   int           count=WHOLE_ARRAY    // number of elements
   );
```

Parameters

dst_array[]

[out] Destination array

src_array[]

[in] Source array

dst_start=0

[in] Starting index from the destination array. By default, start index is 0.

src_start=0

[in] Starting index for the source array. By default, start index is 0.

count=WHOLE_ARRAY

[in] Number of elements that should be copied. By default, the whole array is copied (count=WHOLE_ARRAY).

Return Value

It returns the number of copied elements.

Note

If count<0 or count>src_size-src_start, all the remaining array part is copied. Arrays are copied from left to right. For series arrays, the starting position is correctly defined adjusted for copying from left to right.

If arrays are of different types, during copying it tries to transform each element of a source array into the type of the destination array. A string array can be copied into a string array only. Array of classes and structures containing objects that require initialization aren't copied. An array of structures can be copied into an array of the same type only.

For dynamic arrays with indexing as in timeseries, the size of a destination array is automatically increased to the amount of copied data (if the latter exceeds the array size). The destination array size is not decreased automatically.

Example:

```
#property description "The indicator highlights the candlesticks that are local"
#property description "highs and lows. Interval length for finding"
#property description "extreme values should be found using an input parameters."
```

```
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot
#property indicator_label1  "Extremums"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrLightSteelBlue,clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input int InpNum=4; // Half-interval length
//--- indicator buffers
double ExtOpen[];
double ExtHigh[];
double ExtLow[];
double ExtClose[];
double ExtColor[];
//--- global variables
int    ExtStart=0; // index of the first candlestick that is not an extremum
int    ExtCount=0; // number of non-extremums in the interval
//+------------------------------------------------------------------+
//| Filling out non-extremum candlesticks                            |
//+------------------------------------------------------------------+
void FillCandles(const double &open[],const double &high[],
                 const double &low[],const double &close[])
  {
//--- fill out the candlesticks
   ArrayCopy(ExtOpen,open,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtHigh,high,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtLow,low,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtClose,close,ExtStart,ExtStart,ExtCount);
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtOpen);
   SetIndexBuffer(1,ExtHigh);
   SetIndexBuffer(2,ExtLow);
   SetIndexBuffer(3,ExtClose);
   SetIndexBuffer(4,ExtColor,INDICATOR_COLOR_INDEX);
//--- specify the value, which is not displayed
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- specify the names of indicator buffers for displaying in the data window
```

```
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- set straight indexing in time series
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- variable of the bar calculation start
   int start=prev_calculated;
//--- calculation is not performed for the first InpNum*2 bars
   if(start==0)
     {
      start+=InpNum*2;
      ExtStart=0;
      ExtCount=0;
     }
//--- if the bar has just formed, check the next potential extremum
   if(rates_total-start==1)
      start--;
//--- bar index to be checked for the extremum
   int ext;
//--- indicator value calculation loop
   for(int i=start;i<rates_total-1;i++)
     {
      //--- initially on i bar without drawing
      ExtOpen[i]=0;
      ExtHigh[i]=0;
      ExtLow[i]=0;
      ExtClose[i]=0;
      //--- extremum index for check
      ext=i-InpNum;
      //--- check for the local maximum
      if(IsMax(high,ext))
```

```
         {
          //--- highlight an extremum candlestick
          ExtOpen[ext]=open[ext];
          ExtHigh[ext]=high[ext];
          ExtLow[ext]=low[ext];
          ExtClose[ext]=close[ext];
          ExtColor[ext]=1;
          //--- highlight other candles up to the extremum with a neutral color
          FillCandles(open,high,low,close);
          //--- change the variable colors
          ExtStart=ext+1;
          ExtCount=0;
          //--- pass to the next iteration
          continue;
         }
       //--- check for the local minimum
       if(IsMin(low,ext))
         {
          //--- highlight an extremum candlestick
          ExtOpen[ext]=open[ext];
          ExtHigh[ext]=high[ext];
          ExtLow[ext]=low[ext];
          ExtClose[ext]=close[ext];
          ExtColor[ext]=2;
          //--- highlight other candles up to the extremum with a neutral color
          FillCandles(open,high,low,close);
          //--- change variable values
          ExtStart=ext+1;
          ExtCount=0;
          //--- pass to the next iteration
          continue;
         }
       //--- increase the number of non-extremums at the interval
       ExtCount++;
      }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Check if the current array element is a local high               |
//+------------------------------------------------------------------+
bool IsMax(const double &price[],const int ind)
  {
//--- interval start variable
   int i=ind-InpNum;
//--- interval end period
   int finish=ind+InpNum+1;
//--- check for the first half of the interval
   for(;i<ind;i++)
```

```
     {
      if(price[ind]<=price[i])
         return(false);
     }
//--- check for the second half of the interval
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]<=price[i])
         return(false);
     }
//--- this is an extremum
   return(true);
  }
//+------------------------------------------------------------------+
//| Check if the current array element is a local low                |
//+------------------------------------------------------------------+
bool IsMin(const double &price[],const int ind)
  {
//--- interval start variable
   int i=ind-InpNum;
//--- interval end variable
   int finish=ind+InpNum+1;
//--- check for the first half of the interval
   for(;i<ind;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- check for the second half of the interval
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- this is an extremum
   return(true);
  }
```

ArrayCompare

The function returns the result of comparing two arrays of the same type. It can be used to compare arrays of simple types or custom structures without complex objects, that is the custom structures that do not contain strings, dynamic arrays, classes and other structures with complex objects.

```
int  ArrayCompare(
   const void&   array1[],           // first array
   const void&   array2[],           // second array
   int           start1=0,           // initial offset in the first array
   int           start2=0,           // initial offset in the second array
   int           count=WHOLE_ARRAY    // number of elements for comparison
   );
```

Parameters

array1[]

[in] First array.

array2[]

[in] Second array.

start1=0

[in] The element's initial index in the first array, from which comparison starts. The default start index - 0.

start2=0

[in] The element's initial index in the second array, from which comparison starts. The default start index - 0.

count=WHOLE_ARRAY

[in] Number of elements to be compared. All elements of both arrays participate in comparison by default (count=WHOLE_ARRAY).

Return Value

- -1, if array1[] less than array2[]
- 0, if array1[] equal to array2[]
- 1, if array1[] more than array2[]
- -2, if an error occurs due to incompatibility of the types of compared arrays or if start1, start2 or count values lead to falling outside the array.

Note

The function will not return 0 (the arrays will not be considered equal) if the arrays differ in size and count=WHOLE_ARRAY for the case when one array is a faithful subset of another one. In this case, the result of comparing the sizes of that arrays will be returned: -1, if the size of array1[] is less than the size of array2[] , otherwise 1.

Example:

```
//--- global variables
```

```
double    ExtArrayFirst[];
double    ExtArraySecond[];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the array sizes
   if(ArrayResize(ExtArrayFirst,10)!=10)
     {
      Print("ArrayResize() failed for ExtArrayFirst. Error code: ",GetLastError());
      return;
     }
   if(ArrayResize(ExtArraySecond,10)!=10)
     {
      Print("ArrayResize() failed for ExtArraySecond. Error code: ",GetLastError());
      return;
     }

//--- fill the arrays with the values of i and j indices in a loop
   int total=ArraySize(ExtArrayFirst);
   for(int i=0, j=total-1; i<total; i++,j--)
     {
      //--- fill the ExtArrayFirst array from left to right
      //--- fill the ExtArraySecond array from right to left
      ExtArrayFirst[i]=i;
      ExtArraySecond[i]=j;
     }
//--- compare the arrays and print the result in the log
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
   Result:
   ExtArrayFirst:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   ExtArraySecond:
   9.00000 8.00000 7.00000 6.00000 5.00000 4.00000 3.00000 2.00000 1.00000 0.00000
   Result ArrayCompare(): ExtArrayFirst is smaller than ExtArraySecond (result = -1)
   */

//--- now let's flip the arrays
//--- fill the arrays with the values of i and j indices in a loop
   for(int i=0, j=total-1; i<total; i++,j--)
     {
      //--- fill the ExtArrayFirst array from right to left
      //--- fill the ExtArraySecond array from left to right
      ExtArrayFirst[i]=j;
      ExtArraySecond[i]=i;
     }
```

```
//--- compare the arrays and print the result in the log
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
   Result:
   ExtArrayFirst:
   9.00000 8.00000 7.00000 6.00000 5.00000 4.00000 3.00000 2.00000 1.00000 0.00000
   ExtArraySecond:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   Result ArrayCompare(): ExtArrayFirst is larger than ExtArraySecond (result = 1)
   */

//--- now let's fill the arrays in one direction
//--- fill the arrays with the values of i index in a loop
   for(int i=0; i<total; i++)
     {
      //--- fill both arrays from left to right
      ExtArrayFirst[i]=i;
      ExtArraySecond[i]=i;
     }
//--- compare the arrays and print the result in the log
   ArrayComparePrint(ExtArrayFirst,ExtArraySecond);
   /*
   Result:
   ExtArrayFirst:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   ExtArraySecond:
   0.00000 1.00000 2.00000 3.00000 4.00000 5.00000 6.00000 7.00000 8.00000 9.00000
   Result ArrayCompare(): ExtArrayFirst and ExtArraySecond are equal (result = 0)
   */
  }
//+------------------------------------------------------------------+
//| Compare and display the result                                   |
//+------------------------------------------------------------------+
void ArrayComparePrint(const double &array1[], const double &array2[])
  {
   //--- print the header and contents of the arrays
   Print("ExtArrayFirst:");
   ArrayPrint(array1);
   Print("ExtArraySecond:");
   ArrayPrint(array2);
   //--- compare the arrays and print the comparison result
   int    res=ArrayCompare(array1,array2);
   string res_str=(res>0 ? "ExtArrayFirst is larger than ExtArraySecond" : res<0 ? "Ex
   PrintFormat("Result ArrayCompare(): %s (result = %d)\n",res_str,res);
  }
//+------------------------------------------------------------------+
```

ArrayFree

It frees up a buffer of any dynamic array and sets the size of the zero dimension to 0.

```
void  ArrayFree(
   void&  array[]      // array
   );
```

Parameters

array[]

[in] Dynamic array.

Return Value

No return value.

Note

The need for using ArrayFree() function may not appear too often considering that all used memory is freed at once and main work with the arrays comprises the access to the indicator buffers. The sizes of the buffers are automatically managed by the terminal's executive subsystem.

In case it is necessary to manually manage the memory in complex dynamic environment of the application, ArrayFree() function allows users to free the memory occupied by the already unnecessary dynamic array explicitly and immediately.

Example:

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- predefined constants
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+------------------------------------------------------------------+
//| Dialog class for working with memory                             |
//+------------------------------------------------------------------+
class CMemoryControl : public CAppDialog
  {
private:
   //--- array size
   int               m_arr_size;
   //--- arrays
   char              m_arr_char[];
   int               m_arr_int[];
   float             m_arr_float[];
   double            m_arr_double[];
   long              m_arr_long[];
   //--- labels
   CLabel            m_lbl_memory_physical;
   CLabel            m_lbl_memory_total;
   CLabel            m_lbl_memory_available;
   CLabel            m_lbl_memory_used;
   CLabel            m_lbl_array_size;
   CLabel            m_lbl_array_type;
   CLabel            m_lbl_error;
   CLabel            m_lbl_change_type;
   CLabel            m_lbl_add_size;
   //--- buttons
   CButton           m_button_add;
   CButton           m_button_free;
   //--- combo boxes
   CComboBox         m_combo_box_step;
   CComboBox         m_combo_box_type;
   //--- current value of the array type from the combo box
   int               m_combo_box_type_value;

public:
                     CMemoryControl(void);
                    ~CMemoryControl(void);
   //--- class object creation method
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- handler of chart events
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create labels
   bool              CreateLabel(CLabel &lbl,const string name,const int x,const int
   //--- create control elements
   bool              CreateButton(CButton &button,const string name,const int x,const
   bool              CreateComboBoxStep(void);
   bool              CreateComboBoxType(void);
   //--- event handlers
   void              OnClickButtonAdd(void);
   void              OnClickButtonFree(void);
   void              OnChangeComboBoxType(void);
   //--- methods for working with the current array
   void              CurrentArrayFree(void);
   bool              CurrentArrayAdd(void);
  };
//+------------------------------------------------------------------+
//| Free memory of the current array                                 |
//+------------------------------------------------------------------+
void CMemoryControl::CurrentArrayFree(void)
  {
//--- reset array size
   m_arr_size=0;
```

```
//--- free the array
   if(m_combo_box_type_value==0)
      ArrayFree(m_arr_char);
   if(m_combo_box_type_value==1)
      ArrayFree(m_arr_int);
   if(m_combo_box_type_value==2)
      ArrayFree(m_arr_float);
   if(m_combo_box_type_value==3)
      ArrayFree(m_arr_double);
   if(m_combo_box_type_value==4)
      ArrayFree(m_arr_long);
  }
//+------------------------------------------------------------------+
//| Attempt to add memory for the current array                      |
//+------------------------------------------------------------------+
bool CMemoryControl::CurrentArrayAdd(void)
  {
//--- exit if the size of the used memory exceeds the size of the physical memory
   if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TERMINAL_MEMOR
      return(false);
//--- attempt to allocate memory according to the current type
   if(m_combo_box_type_value==0 && ArrayResize(m_arr_char,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==1 && ArrayResize(m_arr_int,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==2 && ArrayResize(m_arr_float,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==3 && ArrayResize(m_arr_double,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==4 && ArrayResize(m_arr_long,m_arr_size)==-1)
      return(false);
//--- memory allocated
   return(true);
  }
//+------------------------------------------------------------------+
//| Handling events                                                  |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CMemoryControl)
ON_EVENT(ON_CLICK,m_button_add,OnClickButtonAdd)
ON_EVENT(ON_CLICK,m_button_free,OnClickButtonFree)
ON_EVENT(ON_CHANGE,m_combo_box_type,OnChangeComboBoxType)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CMemoryControl::CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CMemoryControl::~CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| Class object creation method                                     |
//+------------------------------------------------------------------+
bool CMemoryControl::Create(const long chart,const string name,const int subwin,
                            const int x1,const int y1,const int x2,const int y2)
  {
//--- create base class object
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- prepare strings for labels
   string str_physical="Memory physical = "+(string)TerminalInfoInteger(TERMINAL_MEMO
   string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_TOT
   string str_available="Memory available = "+(string)TerminalInfoInteger(TERMINAL_ME
   string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED)
//--- create labels
   if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_START+5,str_ph
      return(false);
   if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,str_total,1
      return(false);
   if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_START+55,str
      return(false);
   if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,str_used,12,c
      return(false);
   if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"Array type =
      return(false);
   if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"Array size =
      return(false);
   if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,clrRed))
      return(false);
   if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_START+185,"Chan
      return(false);
   if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,"Add to arr
      return(false);
//--- create control elements
   if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add",12,clrBlue
      return(false);
   if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Free",12,clrB
      return(false);
   if(!CreateComboBoxType())
      return(false);
   if(!CreateComboBoxStep())
      return(false);
//--- initialize the variable
   m_arr_size=0;
```

```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the button                                                |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateButton(CButton &button,const string name,const int x,
                                  const int y,const string str,const int font_size,
                                  const int clr)
  {
//--- create the button
   if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
      return(false);
//--- text
   if(!button.Text(str))
      return(false);
//--- font size
   if(!button.FontSize(font_size))
      return(false);
//--- label color
   if(!button.Color(clr))
      return(false);
//--- add the button to the control elements
   if(!Add(button))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a combo box for the array size                            |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxStep(void)
  {
//--- create the combo box
   if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START+100,Y_START
      return(false);
//--- add elements to the combo box
   if(!m_combo_box_step.ItemAdd("100 000",100000))
      return(false);
   if(!m_combo_box_step.ItemAdd("1 000 000",1000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("10 000 000",10000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("100 000 000",100000000))
      return(false);
//--- set the current combo box element
   if(!m_combo_box_step.SelectByValue(1000000))
      return(false);
//--- add the combo box to control elements
```

```
   if(!Add(m_combo_box_step))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a combo box for the array type                            |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxType(void)
  {
//--- create the combo box
   if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START+100,Y_STAR
      return(false);
//--- add elements to the combo box
   if(!m_combo_box_type.ItemAdd("char",0))
      return(false);
   if(!m_combo_box_type.ItemAdd("int",1))
      return(false);
   if(!m_combo_box_type.ItemAdd("float",2))
      return(false);
   if(!m_combo_box_type.ItemAdd("double",3))
      return(false);
   if(!m_combo_box_type.ItemAdd("long",4))
      return(false);
//--- set the current combo box element
   if(!m_combo_box_type.SelectByValue(3))
      return(false);
//--- store the current combo box element
   m_combo_box_type_value=3;
//--- add the combo box to control elements
   if(!Add(m_combo_box_type))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a label                                                   |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x,
                                 const int y,const string str,const int font_size,
                                 const int clr)
  {
//--- create a label
   if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
      return(false);
//--- text
   if(!lbl.Text(str))
      return(false);
//--- font size
```

```
   if(!lbl.FontSize(font_size))
      return(false);
//--- color
   if(!lbl.Color(clr))
      return(false);
//--- add the label to control elements
   if(!Add(lbl))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Handler of clicking "Add" button event                           |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonAdd(void)
  {
//--- increase the array size
   m_arr_size+=(int)m_combo_box_step.Value();
//--- attempt to allocate memory for the current array
   if(CurrentArrayAdd())
     {
      //--- memory allocated, display the current status on the screen
      m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(T
      m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_ME
      m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
      m_lbl_error.Text("");
     }
   else
     {
      //--- failed to allocate memory, display the error message
      m_lbl_error.Text("Array is too large, error!");
      //--- return the previous array size
      m_arr_size-=(int)m_combo_box_step.Value();
     }
  }
//+------------------------------------------------------------------+
//| Handler of clicking "Free" button event                          |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonFree(void)
  {
//--- free the memory of the current array
   CurrentArrayFree();
//--- display the current status on the screen
   m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERM
   m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMOR
   m_lbl_array_size.Text("Array size = 0");
   m_lbl_error.Text("");
  }
//+------------------------------------------------------------------+
//| Handler of the combo box change event                            |
//+------------------------------------------------------------------+
void CMemoryControl::OnChangeComboBoxType(void)
  {
//--- check if the array's type has changed
   if(m_combo_box_type.Value()!=m_combo_box_type_value)
     {
      //--- free the memory of the current array
      OnClickButtonFree();
      //--- work with another array type
      m_combo_box_type_value=(int)m_combo_box_type.Value();
      //--- display the new array type on the screen
      if(m_combo_box_type_value==0)
         m_lbl_array_type.Text("Array type = char");
      if(m_combo_box_type_value==1)
         m_lbl_array_type.Text("Array type = int");
      if(m_combo_box_type_value==2)
         m_lbl_array_type.Text("Array type = float");
      if(m_combo_box_type_value==3)
         m_lbl_array_type.Text("Array type = double");
      if(m_combo_box_type_value==4)
         m_lbl_array_type.Text("Array type = long");
     }
  }
//--- CMemoryControl class object
CMemoryControl ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create the dialog
   if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
      return(INIT_FAILED);
//--- launch
   ExtDialog.Run();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

ArrayGetAsSeries

It checks direction of an array index.

```
bool  ArrayGetAsSeries(
   const void&  array[]    // array for checking
   );
```

Parameters

array

[in] Checked array.

Return Value

Returns true, if the specified array has the AS_SERIES flag set, i.e. access to the array is performed back to front as in timeseries. A timeseries differs from a usual array in that the indexing of timeseries elements is performed from its end to beginning (from the newest data to old).

Note

To check whether an array belongs to timeseries, use the ArrayIsSeries() function. Arrays of price data passed as input parameters into the OnCalculate() function do not obligatorily have the indexing direction the same as in timeseries. The necessary indexing direction can be set using the ArraySetAsSeries() function.

Example:

```
#property description "Indicator calculates absolute values of the difference between"
#property description "Open and Close or High and Low prices displaying them in a sepa
#property description "as a histrogram."
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- input parameters
input bool InpAsSeries=true; // Indexing direction in the indicator buffer
input bool InpPrices=true;   // Calculation prices (true - Open,Close; false - High,L
//--- indicator buffer
double ExtBuffer[];
//+------------------------------------------------------------------+
//| Calculating indicator values                                     |
//+------------------------------------------------------------------+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                        const double &first[],const double &second[],double &buffer[])
  {
//--- start variable for calculation of bars
   int start=prev_calculated;
```

```
//--- work at the last bar if the indicator values have already been calculated at the
   if(prev_calculated>0)
      start--;
//--- define indexing direction in arrays
   bool as_series_first=ArrayGetAsSeries(first);
   bool as_series_second=ArrayGetAsSeries(second);
   bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- replace indexing direction with direct one if necessary
   if(as_series_first)
      ArraySetAsSeries(first,false);
   if(as_series_second)
      ArraySetAsSeries(second,false);
   if(as_series_buffer)
      ArraySetAsSeries(buffer,false);
//--- calculate indicator values
   for(int i=start;i<rates_total;i++)
      buffer[i]=MathAbs(first[i]-second[i]);
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- bind indicator buffers
   SetIndexBuffer(0,ExtBuffer);
//--- set indexing element in the indicator buffer
   ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- check for what prices the indicator is calculated
   if(InpPrices)
     {
      //--- Open and Close prices
      PlotIndexSetString(0,PLOT_LABEL,"BodySize");
      //--- set the indicator color
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrOrange);
     }
   else
     {
      //--- High and Low prices
      PlotIndexSetString(0,PLOT_LABEL,"ShadowSize");
      //--- set the indicator color
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrDodgerBlue);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                 const int prev_calculated,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//--- calculate the indicator according to the flag value
   if(InpPrices)
      CandleSizeOnBuffer(rates_total,prev_calculated,open,close,ExtBuffer);
   else
      CandleSizeOnBuffer(rates_total,prev_calculated,high,low,ExtBuffer);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries, ArraySetAsSeries

ArrayInitialize

The function initializes a numeric array by a preset value.

For initialization of an array of char type

```
int  ArrayInitialize(
   char      array[],      // initialized array
   char      value         // value that will be set
   );
```

For initialization of an array of short type

```
int  ArrayInitialize(
   short     array[],      // initialized array
   short     value         // value that will be set
   );
```

For initialization of an array of int type

```
int  ArrayInitialize(
   int       array[],      // initialized array
   int       value         // value that will be set
   );
```

For initialization of an array of long type

```
int  ArrayInitialize(
   long      array[],      // initialized array
   long      value         // value that will be set
   );
```

For initialization of an array of float type

```
int  ArrayInitialize(
   float     array[],      // initialized array
   float     value         // value that will be set
   );
```

For initialization of an array of double type

```
int  ArrayInitialize(
   double    array[],      // initialized array
   double    value         // value that will be set
   );
```

For initialization of an array of bool type

```
int  ArrayInitialize(
   bool      array[],      // initialized array
   bool      value         // value that will be set
   );
```

For initialization of an array of uint type

```
int  ArrayInitialize(
   uint     array[],      // initialized array
   uint     value         // value that will be set
   );
```

Parameters

array[]

[out] Numeric array that should be initialized.

value

[in] New value that should be set to all array elements.

Return Value

Number of initialized elements.

Note

The ArrayResize() function allows to set size of an array with a reserve for further expansion without the physical relocation of memory. It is implemented for the better performance, because the operations of memory relocation are reasonably slow.

Initialization of the array using ArrayInitialize(array, init_val) doesn't mean the initialization with the same value of reserve elements allocated for this array. At further expanding of the *array* using the ArrayResize() function, the elements will be added at the end of the array, their values will be undefined and in most cases will not be equal to *init_value*.

Example:

```
void OnStart()
  {
//--- dynamic array
   double array[];
//--- let's set the array size for 100 elements and reserve a buffer for another 10 el
   ArrayResize(array,100,10);
//--- initialize the array elements with EMPTY_VALUE=DBL_MAX value
   ArrayInitialize(array,EMPTY_VALUE);
   Print("Values of 10 last elements after initialization");
   for(int i=90;i<100;i++) printf("array[%d] = %G",i,array[i]);
//--- expand the array by 5 elements
   ArrayResize(array,105);
   Print("Values of 10 last elements after ArrayResize(array,105)");
//--- values of 5 last elements are obtained from reserve buffer
   for(int i=95;i<105;i++) printf("array[%d] = %G",i,array[i]);
  }
```

ArrayFill

The function fills an array with the specified value.

```
void  ArrayFill(
   void&   array[],       // array
   int     start,          // starting index
   int     count,          // number of elements to fill
   void    value           // value
   );
```

Parameters

array[]

[out] Array of simple type (char, uchar, short, ushort, int, uint, long, ulong, bool, color, datetime, float, double).

start

[in] Starting index. In such a case, specified AS_SERIES flag is ignored.

count

[in] Number of elements to fill.

value

[in] Value to fill the array with.

Return Value

No return value.

Note

When ArrayFill() function is called, normal indexation direction (from left to right) is always implied. It means that the change of the order of access to the array elements using ArraySetAsSeries() function is ignored.

A multidimensional array is shown as one-dimensional when processed by ArrayFill() function. For example, array[2][4] is processed as array[8]. Therefore, you may specify the initial element's index to be equal to 5 when working with this array. Thus, the call of ArrayFill(array, 5, 2, 3.14) for array[2][4] fills array[1][1] and array[1][2] elements with 3.14.

Example:

```
void OnStart()
  {
//--- declare dynamic array
   int a[];
//--- set size
   ArrayResize(a,10);
//--- fill first 5 elements with 123
   ArrayFill(a,0,5,123);
//--- fill next 5 elements with 456
   ArrayFill(a,5,5,456);
//--- show values
   for(int i=0;i<ArraySize(a);i++) printf("a[%d] = %d",i,a[i]);
  }
```

ArrayIsDynamic

The function checks whether an array is dynamic.

```
bool  ArrayIsDynamic(
   const void&  array[]     // checked array
   );
```

Parameters

array[]

[in] Checked array.

Return Value

It returns true if the selected array is dynamic, otherwise it returns false.

Example:

```
#property description "This indicator does not calculate values. It makes a single att
#property description "apply the call of ArrayFree() function to three arrays: dynamic
#property description "an indicator buffer. Results are shown in Experts journal."
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- global variables
double ExtDynamic[];   // dynamic array
double ExtStatic[100]; // static array
bool   ExtFlag=true;   // flag
double ExtBuff[];      // indicator buffer
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- allocate memory for the array
   ArrayResize(ExtDynamic,100);
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtBuff);
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])
  {
//--- perform a single analysis
   if(ExtFlag)
     {
      //--- attempt to free memory for arrays
      //--- 1. Dynamic array
      Print("+============================+");
      Print("1. Check dynamic array:");
      Print("Size before memory is freed = ",ArraySize(ExtDynamic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtDynamic) ? "Yes" : "No");
      //--- attempt to free array memory
      ArrayFree(ExtDynamic);
      Print("Size after memory is freed = ",ArraySize(ExtDynamic));
      //--- 2. Static array
      Print("2. Check static array:");
      Print("Size before memory is freed = ",ArraySize(ExtStatic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtStatic) ? "Yes" : "No");
      //--- attempt to free array memory
      ArrayFree(ExtStatic);
      Print("Size after memory is freed = ",ArraySize(ExtStatic));
      //--- 3. Indicator buffer
      Print("3. Check indicator buffer:");
      Print("Size before memory is freed = ",ArraySize(ExtBuff));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtBuff) ? "Yes" : "No");
      //--- attempt to free array memory
      ArrayFree(ExtBuff);
      Print("Size after memory is freed = ",ArraySize(ExtBuff));
      //--- change the flag value
      ExtFlag=false;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Access to timeseries and indicators]()

ArrayIsSeries

The function checks whether an array is a timeseries.

```
bool  ArrayIsSeries(
   const void&  array[]     // checked array
   );
```

Parameters

array[]

[in] Checked array.

Return Value

It returns true, if a checked array is an array timeseries, otherwise it returns false. Arrays passed as a parameter to the OnCalculate() function must be checked for the order of accessing the array elements by ArrayGetAsSeries().

Example:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         Label1Buffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                  const long &tick_volume[],
                  const long &volume[],
                  const int &spread[])
  {
//---
   if(ArrayIsSeries(open))
      Print("open[] is timeseries");
   else
      Print("open[] is not timeseries!!!");
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Access to timeseries and indicators]

ArrayMaximum

Searches for the largest element in the first dimension of a multidimensional numeric array.

```
int  ArrayMaximum(
   const void&    array[],            // array for search
   int            start=0,            // index to start checking with
   int            count=WHOLE_ARRAY   // number of checked elements
   );
```

Parameters

array[]

[in] A numeric array, in which search is made.

start=0

[in] Index to start checking with.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=WHOLE_ARRAY).

Return Value

The function returns an index of a found element taking into account the array serial. In case of failure it returns -1.

Note

The AS_SERIES flag value is taken into account while searching for a maximum.

Functions ArrayMaximum and ArrayMinimum accept any-dimensional arrays as a parameter. However, searching is always applied to the first (zero) dimension.

Example:

```
#property description "The indicator displays larger timeframe's candlesticks on the c
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- plot 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
```

```
//---- plot 5
#property indicator_label5  "BullBody"
#property indicator_color5  clrOlive,clrOlive
//---- plot 6
#property indicator_label6  "BullBodyEnd"
#property indicator_color6  clrOlive,clrOlive
//---- plot 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // Time frame for the indicato
input datetime        InpDateStart=D'2013.01.01 00:00'; // Analysis start date
//--- indicator buffers for bearish candlesticks
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- indicator buffers for bullish candlesticks
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- global variables
datetime ExtTimeBuff[];      // larger time frame's time buffer
int      ExtSize=0;          // time buffer size
int      ExtCount=0;         // index inside time buffer
int      ExtStartPos=0;      // initial position for the indicator calculation
bool     ExtStartFlag=true;  // auxiliary flag for receiving the initial position
datetime ExtCurrentTime[1];  // last time of the larger time frame's bar generation
datetime ExtLastTime;        // last time from the larger time frame, for which the ca
bool     ExtBearFlag=true;   // flag for defining the order of writing the data to bea
bool     ExtBullFlag=true;   // flag for defining the order of writing the data to bul
int      ExtIndexMax=0;      // index of the maximum element in the array
int      ExtIndexMin=0;      // index of the minimum element in the array
int      ExtDirectionFlag=0; // price movement direction for the current candlestick
//--- shift between the candlestick's open and close price for correct drawing
```

```
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
//| Filling the basic part of the candlestick                        |
//+------------------------------------------------------------------+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- find the index of the maximum and minimum elements in the arrays
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // maximum in High
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // minimum in Low
//--- define how many bars from the current time frame are to be filled out
   int count=fill_index-start+1;
//--- if the close price at the first bar exceeds the one at the last bar, the candles
   if(open[start]>close[last])
     {
      //--- if the candlestick has been bullish before that, clear the values of bulli
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShac
      //--- bearish candlestick
      ExtDirectionFlag=-1;
      //--- generate the candlestick
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShac
                     close[last],high[index_max],low[index_min],start,count,ExtBearFla
      //--- exit the function
      return;
     }
//--- if the close price at the first bar is less than the one at the last bar, the ca
   if(open[start]<close[last])
     {
      //--- if the candlestick has been bearish before that, clear the values of beari
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShac
      //--- bullish candlestick
      ExtDirectionFlag=1;
      //--- generate the candlestick
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShac
                     open[start],high[index_max],low[index_min],start,count,ExtBullFla
      //--- exit the function
      return;
     }
//--- if you are in this part of the function, the open price at the first bar is equa
//--- the close price at the last bar; such candlestick is considered bearish
//--- if the candlestick has been bullish before that, clear the values of bullish inc
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowS
//--- bearish candlestick
   ExtDirectionFlag=-1;
```

```
//--- if close and open prices are equal, use the shift for correct display
   if(high[index_max]!=low[index_min])
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShac
                     open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShac
                     open[start],open[start]-ExtEmptyBodySize,high[index_max],
                     high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| Fill out the end of the candlestick                              |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- do not draw in case of a single bar
   if(last-start==0)
      return;
//--- if the close price at the first bar exceeds the one at the last bar, the candles
   if(open[start]>close[last])
     {
      //--- generate the end of the candlestick
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start],close[last],high[index_max],low[index_min],fill_index,
      //--- exit the function
      return;
     }
//--- if the close price at the first bar is less than the one at the last bar, the ca
   if(open[start]<close[last])
     {
      //--- generate the end of the candlestick
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ext
                    close[last],open[start],high[index_max],low[index_min],fill_index,
      //--- exit the function
      return;
     }
//--- if you are in this part of the function, the open price at the first bar is equa
//--- the close price at the last bar; such candlestick is considered bearish
//--- generate the end of the candlestick
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_i
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEr
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check the indicator period
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- display price data in the foreground
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- binding indicator buffers
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- set some property values for creating the indicator
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // graphical construction ty
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // drawing line style
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // drawing line width
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- in case there are no calculated bars yet
   if(prev_calculated==0)
     {
      //--- receive larger time frame's bars arrival time
      if(!GetTimeData())
         return(0);
     }
//--- set direct indexing
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- start variable for calculation of bars
   int start=prev_calculated;
//--- if the bar is generated, recalculate the indicator value on it
   if(start!=0 && start==rates_total)
      start--;
//--- the loop for calculating the indicator values
   for(int i=start;i<rates_total;i++)
     {
      //--- fill i elements of the indicator buffers by empty values
      FillIndicatorBuffers(i);
      //--- perform calculation for bars starting from InpDateStart date
      if(time[i]>=InpDateStart)
        {
         //--- define position, from which the values are to be displayed, for the fi
         if(ExtStartFlag)
           {
            //--- store the number of the initial bar
            ExtStartPos=i;
            //--- define the first date from the larger time frame exceeding time[i]
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- change the value of the flag in order not to run into this block aga
            ExtStartFlag=false;
           }
         //--- check if there are still any elements in the array
         if(ExtCount<ExtSize)
           {
            //--- wait for the current time frame's value to reach the larger time fra
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //--- draw the main part of the candlestick (without filling out the ar
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,ExtI
               //--- fill out the end of the candlestick (the area between the last ar
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
               //--- shift the initial position for drawing the next candlestick
               ExtStartPos=i;
               //--- increase the array counter
               ExtCount++;
              }
            else
               continue;
           }
         else
           {
            //--- reset the array values
            ResetLastError();
            //--- receive the last date from the larger time frame
            if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
              {
               Print("Data copy error, code = ",GetLastError());
               return(0);
              }
            //--- if the new date is later, stop generating the candlestick
            if(ExtCurrentTime[0]>ExtLastTime)
              {
               //--- clear the area between the last and penultimate bars in the main
               ClearEndOfBodyMain(i-1);
               //--- fill out the area using auxiliary indicator buffers
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
               //--- shift the initial position for drawing the next candlestick
               ExtStartPos=i;
               //--- reset price direction flag
               ExtDirectionFlag=0;
               //--- store the new last date
               ExtLastTime=ExtCurrentTime[0];
              }
            else
              {
               //--- generate the candlestick
               FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtInde
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Check correctness of the specified indicator period              |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
//--- the indicator period should exceed the timeframe on which it is displayed
```

```
   if(current_period>=high_period)
     {
      Print("Error! The value of the indicator period should exceed the value of the
      return(false);
     }
//--- if the indicator period is one week or month, the period is correct
   if(high_period>32768)
      return(true);
//--- convert period values to minutes
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- the indicator period should be multiple of the time frame it is displayed on
   if(high_period%current_period!=0)
     {
      Print("Error! The value of the indicator period should be multiple of the value
      return(false);
     }
//--- the indicator period should exceed the time frame it is displayed on 3 or more
   if(high_period/current_period<3)
     {
      Print("Error! The indicator period should exceed the current time frame 3 or mo
      return(false);
     }
//--- the indicator period is correct for the current time frame
   return(true);
  }
//+------------------------------------------------------------------+
//| Receive time data from the larger time frame                     |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- reset the error value
   ResetLastError();
//--- copy all data for the current time
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- receive the error code
      int code=GetLastError();
      //--- print out the error message
      PrintFormat("Data copy error! %s",code==4401
                  ? "History is still being uploaded!"
                  : "Code = "+IntegerToString(code));
      //--- return false to make a repeated attempt to download data
      return(false);
     }
//--- receive the array size
   ExtSize=ArraySize(ExtTimeBuff);
```

```
//--- set the loop index for the array to zero
   ExtCount=0;
//--- set the current candlestick's position on the time frame to zero
   ExtStartPos=0;
   ExtStartFlag=true;
//--- store the last time value from the larger time frame
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Function forms the main part of the candlestick. Depending on the flag's |
//| value, the function defines what data and arrays are                |
//| to be used for correct display.                                      |
//+------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- check the flag's value
   if(flag)
     {
      //--- generate the candlestick's body
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- generate the candlestick's shadow
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- generate the candlestick's body
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- generate the candlestick's shadow
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+------------------------------------------------------------------------+
//| The function forms the end of the candlestick. Depending on the flag's value, |
//| the function defines what data and arrays are                               |
//| to be used for correct display.                                             |
//+------------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
                   const int end,bool &flag)
  {
//--- check the flag's value
```

```
   if(flag)
     {
      //--- generate the end of the candlestick's body
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- generate the end of the candlestick's shadow
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- change the flag's value to the opposite one
      flag=false;
     }
   else
     {
      //--- generate the end of the candlestick's body
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- generate the end of the candlestick's shadow
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- change the flag's value to the opposite one
      flag=true;
     }
  }
//+---------------------------------------------------------------------------------+
//| Clear the end of the candlestick (the area between the last and the penultimate |
//| bar)                                                                            |
//+---------------------------------------------------------------------------------+
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
  }
//+--------------------------------------------------------------------------+
//| Clear the candlestick                                                    |
//+--------------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- check
   if(count!=0)
     {
      //--- fill indicator buffers with empty values
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+--------------------------------------------------------------------------+
//| Generate the main part of the candlestick                                |
//+--------------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- check
   if(count!=0)
     {
      //--- fill indicator buffers with values
      ArrayFill(fst,start,count,fst_value);
      ArrayFill(snd,start,count,snd_value);
     }
  }
//+------------------------------------------------------------------------------+
//| Generate the end of the candlestick                                          |
//+------------------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
  {
//--- fill indicator buffers with values
   ArrayFill(fst,last-1,2,fst_value);
   ArrayFill(snd,last-1,2,snd_value);
  }
//+------------------------------------------------------------------+
//| Fill i element of the indicator buffers by empty values          |
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
  {
//--- set an empty value in the cell of the indicator buffers
   ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
  }
```

ArrayMinimum

Searches for the lowest element in the first dimension of a multidimensional numeric array.

```
int  ArrayMinimum(
   const void&    array[],            // array for search
   int            start=0,            // index to start checking with
   int            count=WHOLE_ARRAY   // number of checked elements
   );
```

Parameters

array[]

[in] A numeric array, in which search is made.

start=0

[in] Index to start checking with.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=WHOLE_ARRAY).

Return Value

The function returns an index of a found element taking into account the array serial. In case of failure it returns -1.

Note

The AS_SERIES flag value is taken into account while searching for a minimum.

Functions ArrayMaximum and ArrayMinimum accept any-dimensional arrays as a parameter. However, searching is always applied to the first (zero) dimension.

Example:

```
#property description "The indicator displays larger timeframe's candlesticks on the
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- plot 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
```

```
//---- plot 5
#property indicator_label5  "BullBody"
#property indicator_color5  clrOlive,clrOlive
//---- plot 6
#property indicator_label6  "BullBodyEnd"
#property indicator_color6  clrOlive,clrOlive
//---- plot 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // Time frame for the indicat
input datetime        InpDateStart=D'2013.01.01 00:00'; // Analysis start date
//--- indicator buffers for bearish candlesticks
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- indicator buffers for bullish candlesticks
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- global variables
datetime ExtTimeBuff[];      // larger time frame's time buffer
int      ExtSize=0;          // time buffer size
int      ExtCount=0;         // index inside time buffer
int      ExtStartPos=0;      // initial position for the indicator calculation
bool     ExtStartFlag=true;  // auxiliary flag for receiving the initial position
datetime ExtCurrentTime[1];  // last time of the larger time frame's bar generation
datetime ExtLastTime;        // last time from the larger time frame, for which the ca
bool     ExtBearFlag=true;   // flag for defining the order of writing the data to bea
bool     ExtBullFlag=true;   // flag for defining the order of writing the data to bul
int      ExtIndexMax=0;      // index of the maximum element in the array
int      ExtIndexMin=0;      // index of the minimum element in the array
int      ExtDirectionFlag=0; // price movement direction for the current candlestick
//--- shift between the candlestick's open and close price for correct drawing
```

```
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
//| Filling the basic part of the candlestick                        |
//+------------------------------------------------------------------+
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- find the index of the maximum and minimum elements in the arrays
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // maximum in High
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // minimum in Low
//--- define how many bars from the current time frame are to be filled out
   int count=fill_index-start+1;
//--- if the close price at the first bar exceeds the one at the last bar, the candles
   if(open[start]>close[last])
     {
      //--- if the candlestick has been bullish before that, clear the values of bulli
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
      //--- bearish candlestick
      ExtDirectionFlag=-1;
      //--- generate the candlestick
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     close[last],high[index_max],low[index_min],start,count,ExtBearFla
      //--- exit the function
      return;
     }
//--- if the close price at the first bar is less than the one at the last bar, the ca
   if(open[start]<close[last])
     {
      //--- if the candlestick has been bearish before that, clear the values of beari
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
      //--- bullish candlestick
      ExtDirectionFlag=1;
      //--- generate the candlestick
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShad
                     open[start],high[index_max],low[index_min],start,count,ExtBullFla
      //--- exit the function
      return;
     }
//--- if you are in this part of the function, the open price at the first bar is equa
//--- the close price at the last bar; such candlestick is considered bearish
//--- if the candlestick has been bullish before that, clear the values of bullish ind
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowS
//--- bearish candlestick
   ExtDirectionFlag=-1;
```

```
//--- if close and open prices are equal, use the shift for correct display
   if(high[index_max]!=low[index_min])
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShad
                     open[start],open[start]-ExtEmptyBodySize,high[index_max],
                     high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| Fill out the end of the candlestick                              |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- do not draw in case of a single bar
   if(last-start==0)
      return;
//--- if the close price at the first bar exceeds the one at the last bar, the candles
   if(open[start]>close[last])
     {
      //--- generate the end of the candlestick
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start],close[last],high[index_max],low[index_min],fill_index,
      //--- exit the function
      return;
     }
//--- if the close price at the first bar is less than the one at the last bar, the ca
   if(open[start]<close[last])
     {
      //--- generate the end of the candlestick
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ext
                    close[last],open[start],high[index_max],low[index_min],fill_index,
      //--- exit the function
      return;
     }
//--- if you are in this part of the function, the open price at the first bar is equa
//--- the close price at the last bar; such candlestick is considered bearish
//--- generate the end of the candlestick
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ext
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEm
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check the indicator period
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- display price data in the foreground
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- binding indicator buffers
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- set some property values for creating the indicator
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // graphical construction ty
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // drawing line style
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // drawing line width
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- in case there are no calculated bars yet
   if(prev_calculated==0)
     {
      //--- receive larger time frame's bars arrival time
      if(!GetTimeData())
         return(0);
     }
//--- set direct indexing
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- start variable for calculation of bars
   int start=prev_calculated;
//--- if the bar is generated, recalculate the indicator value on it
   if(start!=0 && start==rates_total)
      start--;
//--- the loop for calculating the indicator values
   for(int i=start;i<rates_total;i++)
     {
      //--- fill i elements of the indicator buffers by empty values
      FillIndicatorBuffers(i);
      //--- perform calculation for bars starting from InpDateStart date
      if(time[i]>=InpDateStart)
        {
         //--- define position, from which the values are to be displayed, for the fi
         if(ExtStartFlag)
           {
            //--- store the number of the initial bar
            ExtStartPos=i;
            //--- define the first date from the larger time frame exceeding time[i]
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- change the value of the flag in order not to run into this block aga
            ExtStartFlag=false;
           }
         //--- check if there are still any elements in the array
         if(ExtCount<ExtSize)
           {
            //--- wait for the current time frame's value to reach the larger time fra
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //--- draw the main part of the candlestick (without filling out the a
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,ExtI
               //--- fill out the end of the candlestick (the area between the last a
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
               //--- shift the initial position for drawing the next candlestick
               ExtStartPos=i;
               //--- increase the array counter
               ExtCount++;
              }
            else
               continue;
           }
         else
           {
            //--- reset the array values
            ResetLastError();
            //--- receive the last date from the larger time frame
            if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
              {
               Print("Data copy error, code = ",GetLastError());
               return(0);
              }
            //--- if the new date is later, stop generating the candlestick
            if(ExtCurrentTime[0]>ExtLastTime)
              {
               //--- clear the area between the last and penultimate bars in the main
               ClearEndOfBodyMain(i-1);
               //--- fill out the area using auxiliary indicator buffers
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIn
               //--- shift the initial position for drawing the next candlestick
               ExtStartPos=i;
               //--- reset price direction flag
               ExtDirectionFlag=0;
               //--- store the new last date
               ExtLastTime=ExtCurrentTime[0];
              }
            else
              {
               //--- generate the candlestick
               FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndex
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Check correctness of the specified indicator period              |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
//--- the indicator period should exceed the timeframe on which it is displayed
```

```
   if(current_period>=high_period)
     {
      Print("Error! The value of the indicator period should exceed the value of the
      return(false);
     }
//--- if the indicator period is one week or month, the period is correct
   if(high_period>32768)
      return(true);
//--- convert period values to minutes
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- the indicator period should be multiple of the time frame it is displayed on
   if(high_period%current_period!=0)
     {
      Print("Error! The value of the indicator period should be multiple of the value
      return(false);
     }
//--- the indicator period should exceed the time frame it is displayed on 3 or more
   if(high_period/current_period<3)
     {
      Print("Error! The indicator period should exceed the current time frame 3 or mo
      return(false);
     }
//--- the indicator period is correct for the current time frame
   return(true);
  }
//+------------------------------------------------------------------+
//| Receive time data from the larger time frame                     |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- reset the error value
   ResetLastError();
//--- copy all data for the current time
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- receive the error code
      int code=GetLastError();
      //--- print out the error message
      PrintFormat("Data copy error! %s",code==4401
                  ? "History is still being uploaded!"
                  : "Code = "+IntegerToString(code));
      //--- return false to make a repeated attempt to download data
      return(false);
     }
//--- receive the array size
   ExtSize=ArraySize(ExtTimeBuff);
```

```
//--- set the loop index for the array to zero
   ExtCount=0;
//--- set the current candlestick's position on the time frame to zero
   ExtStartPos=0;
   ExtStartFlag=true;
//--- store the last time value from the larger time frame
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------------+
//| Function forms the main part of the candlestick. Depending on the flag's |
//| value, the function defines what data and arrays are                     |
//| to be used for correct display.                                          |
//+--------------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- check the flag's value
   if(flag)
     {
      //--- generate the candlestick's body
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- generate the candlestick's shadow
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- generate the candlestick's body
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- generate the candlestick's shadow
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+--------------------------------------------------------------------------------+
//| The function forms the end of the candlestick. Depending on the flag's value,  |
//| the function defines what data and arrays are                                  |
//| to be used for correct display.                                                |
//+--------------------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
                   const int end,bool &flag)
  {
//--- check the flag's value
```

```
   if(flag)
     {
      //--- generate the end of the candlestick's body
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- generate the end of the candlestick's shadow
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- change the flag's value to the opposite one
      flag=false;
     }
   else
     {
      //--- generate the end of the candlestick's body
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- generate the end of the candlestick's shadow
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- change the flag's value to the opposite one
      flag=true;
     }
  }
//+-----------------------------------------------------------------------------------
//| Clear the end of the candlestick (the area between the last and the penultimate
//| bar)
//+-----------------------------------------------------------------------------------
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
  }
//+------------------------------------------------------------------+
//| Clear the candlestick                                            |
//+------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- check
   if(count!=0)
     {
      //--- fill indicator buffers with empty values
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+------------------------------------------------------------------+
//| Generate the main part of the candlestick                        |
//+------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- check
   if(count!=0)
     {
      //--- fill indicator buffers with values
      ArrayFill(fst,start,count,fst_value);
      ArrayFill(snd,start,count,snd_value);
     }
  }
//+------------------------------------------------------------------+
//| Generate the end of the candlestick                              |
//+------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
  {
//--- fill indicator buffers with values
   ArrayFill(fst,last-1,2,fst_value);
   ArrayFill(snd,last-1,2,snd_value);
  }
//+------------------------------------------------------------------+
//| Fill i element of the indicator buffers by empty values          |
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
  {
//--- set an empty value in the cell of the indicator buffers
   ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
  }
```

ArrayPrint

Prints an array of a simple type or a simple structure into journal.

```
void  ArrayPrint(
   const void&    array[],             // printed array
   uint           digits=_Digits,      // number of decimal places
   const string   separator=NULL,      // separator of the structure field values
   ulong          start=0,             // first printed element index
   ulong          count=WHOLE_ARRAY,   // number of printed elements
   ulong          flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_
   );
```

Parameters

array[]

[in] Array of a simple type or a simple structure.

digits=_Digits

[in] The number of decimal places for real types. The default value is _Digits.

separator=NULL

[in] Separator of the structure element field values. The default value NULL means an empty line. A space is used as a separator in that case.

start=0

[in] The index of the first printed array element. It is printed from the zero index by default.

count=WHOLE_ARRAY

[in] Number of the array elements to be printed. The entire array is displayed by default (count=WHOLE_ARRAY).

flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_ALIGN

[in] Combination of flags setting the output mode. All flags are enabled by default:

- ☐ARRAYPRINT_HEADER – print headers for the structure array
- ☐ARRAYPRINT_INDEX – print index at the left side
- ☐ARRAYPRINT_LIMIT – print only the first 100 and the last 100 array elements. Use if you want to print only a part of a large array.
- ☐ARRAYPRINT_ALIGN – enable alignment of the printed values – numbers are aligned to the right, while lines to the left.
- ☐ARRAYPRINT_DATE – when printing datetime, print the date in the dd.mm.yyyy format
- ☐ARRAYPRINT_MINUTES – when printing datetime, print the time in the HH:MM format
- ☐ARRAYPRINT_SECONDS – when printing datetime, print the time in the HH:MM:SS format

Return Value

No

Note

ArrayPrint() does not print all structure array fields into journal – array and object pointer fields are skipped. These columns are simply not printed for more convenient presentation. If you need to print all structure fields, you need to write your own mass print function with the desired formatting.

Example:

```
//--- print the values of the last 10 bars
   MqlRates rates[];
   if(CopyRates(_Symbol,_Period,1,10,rates))
     {
      ArrayPrint(rates);
      Print("Check\n[time]\t[open]\t[high]\t[low]\t[close]\t[tick_volume]\t[spread]\t
      for(int i=0;i<10;i++)
        {
         PrintFormat("[%d]\t%s\t%G\t%G\t%G\t%G\t%G\t%G\t%I64d\t",i,
         TimeToString(rates[i].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS),
         rates[i].open,rates[i].high,rates[i].low,rates[i].close,
         rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
     }
   else
      PrintFormat("CopyRates failed, error code=%d",GetLastError());
//--- example of printing
/*
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295         18110       10   1
   [1] 2016.11.09 05:00:00 1.12296 1.12825 1.11930 1.12747         17829        9   1
   [2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744         13458       10
   [3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194         15362        9   1
   [4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172         16833        9   1
   [5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052         15933        8   1
   [6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528         11888        9
   [7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915          7284       10
   [8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904          8710        9
   [9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263          8956        7
   Check
   [time]  [open]  [high]  [low]  [close]  [tick_volume]  [spread]  [real_volume]
   [0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295 18110 10 17300175000
   [1] 2016.11.09 05:00:00 1.12296 1.12825 1.1193 1.12747 17829 9 15632176000
   [2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744 13458 10 9593492000
   [3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194 15362 9 12352245000
   [4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172 16833 9 12961333000
   [5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052 15933 8 10720384000
   [6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528 11888 9 8084811000
   [7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915 7284 10 5087113000
   [8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904 8710 9 6769629000
   [9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263 8956 7 7192138000
*/
```

See also

FileSave, FileLoad

ArrayRange

The function returns the number of elements in a selected array dimension.

```
int  ArrayRange(
   const void&    array[],       // array for check
   int            rank_index     // index of dimension
   );
```

Parameters

array[]

[in] Checked array.

rank_index

[in] Index of dimension.

Return Value

Number of elements in a selected array dimension.

Note

Since indexes start at zero, the number of the array dimensions is one greater than the index of the last dimension.

Example:

```
void OnStart()
  {
//--- create four-dimensional array
   double array[][5][2][4];
//--- set the size of the zero dimension
   ArrayResize(array,10,10);
//--- print dimensions
   int temp;
   for(int i=0;i<4;i++)
     {
      //--- receive the size of i dimension
      temp=ArrayRange(array,i);
      //--- print
      PrintFormat("dim = %d, range = %d",i,temp);
     }
//--- Result
// dim = 0, range = 10
// dim = 1, range = 5
// dim = 2, range = 2
// dim = 3, range = 4
  }
```

ArrayResize

The function sets a new size for the first dimension

```
int  ArrayResize(
   void&  array[],              // array passed by reference
   int    new_size,             // new array size
   int    reserve_size=0        // reserve size value (excess)
   );
```

Parameters

array[]

[out] Array for changing sizes.

new_size

[in] New size for the first dimension.

reserve_size=0

[in] Distributed size to get reserve.

Return Value

If executed successfully, it returns count of all elements contained in the array after resizing, otherwise, returns -1, and array is not resized.

If ArrayResize() is applied to a static array, a timeseries or an indicator buffer, the array size remains the same - these arrays will not be reallocated. In this case, if *new_size*<=ArraySize(*array*), the function will only return *new_size*; otherwise a value of -1 will be returned.

Note

The function can be applied only to dynamic arrays. It should be noted that you cannot change the size of dynamic arrays assigned as indicator buffers by the SetIndexBuffer() function. For indicator buffers, all operations of resizing are performed by the runtime subsystem of the terminal.

Total amount of elements in the array cannot exceed 2147483647.

With the frequent memory allocation, it is recommended to use a third parameter that sets a reserve to reduce the number of physical memory allocations. All the subsequent calls of ArrayResize do not lead to physical reallocation of memory, but only change the size of the first array dimension within the reserved memory. It should be remembered that the third parameter will be used only during physical memory allocation. For example:

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
   ArrayResize(arr,i,1000);
```

In this case the memory will be reallocated twice, first before entering the 3000 iterations loop (the array size will be set to 1000), and the second time with i equal to 2000. If we skip the third parameter, there will be 2000 physical reallocations of memory, which will slow down the program.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Counters
   ulong start=GetTickCount();
   ulong now;
   int   count=0;
//--- An array for demonstration of a quick version
   double arr[];
   ArrayResize(arr,100000,100000);
//--- Check how fast the variant with memory reservation works
   Print("--- Test Fast: ArrayResize(arr,100000,100000)");
   for(int i=1;i<=300000;i++)
     {
      //--- Set a new array size specifying the reserve of 100,000 elements!
      ArrayResize(arr,i,100000);
      //--- When reaching a round number, show the array size and the time spent
      if(ArraySize(arr)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(arr),(now-sta
         start=now;
        }
     }
//--- Now show, how slow the version without memory reservation is
   double slow[];
   ArrayResize(slow,100000,100000);
//---
   count=0;
   start=GetTickCount();
   Print("---- Test Slow: ArrayResize(slow,100000)");
//---
   for(int i=1;i<=300000;i++)
     {
      //--- Set a new array size, but without the additional reserve
      ArrayResize(slow,i);
      //--- When reaching a round number, show the array size and the time spent
      if(ArraySize(slow)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(slow)=%d Time=%d ms",count,ArraySize(slow),(now-st
         start=now;
        }
     }
  }
//--- A sample result of the script
```

```
/*
   Test_ArrayResize (EURUSD,H1)   --- Test Fast: ArrayResize(arr,100000,100000)
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(arr)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(arr)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(arr)=300000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   ---- Test Slow: ArrayResize(slow,100000)
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(slow)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(slow)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(slow)=300000 Time=228511 ms
*/
```

See also

ArrayInitialize

ArrayInsert

Inserts the specified number of elements from a source array to a receiving one starting from a specified index.

```
bool  ArrayInsert(
   void&         dst_array[],          // receiving array
   const void&   src_array[],          // source array
   uint          dst_start,            // receiver array index to be inserted
   uint          src_start=0,          // source array index to be copied
   uint          count=WHOLE_ARRAY     // number of elements to insert
   );
```

Parameters

dst_array[]

[in][out] Receiving array the elements should be added to.

src_array[]

[in] Source array the elements are to be added from.

dst_start

[in] Index in the receiving array for inserting elements from the source array.

src_start=0

[in] Index in the source array, starting from which the elements of the source array are taken for insertion.

count

[in] Number of elements to be added from the source array. The WHOLE_ARRAY means all elements from the specified index up to the end of the array.

Return Value

Returns true if successful, otherwise - false. To get information about the error, call the GetLastError() function. Possible errors:

- 5052 – ERR_SMALL_ARRAY (the *start* and/or *count* parameters are set incorrectly or the *src_array[]* source array is empty),
- 5056 – ERR_SERIES_ARRAY (the array cannot be changed, indicator buffer),
- 4006 – ERR_INVALID_ARRAY (copying to oneself is not allowed, or the arrays are of different types, or there is a fixed-size array containing class objects or destructor structures),
- 4005 - ERR_STRUCT_WITHOBJECTS_ORCLASS (the array contains no POD structures meaning a simple copying is impossible),
- Errors occurred when changing the *dst_array[]* receiving array size are provided in the ArrayRemove() function description.

Note

If the function is used for a fixed-size array, the size of the *dst_array[]* receiving array itself does not change. Starting from the *dst_start* position, the elements of the receiving array are shifted to the right (the last *counts* of the elements "come off"), while the elements copied from the source array take their place.

You cannot insert the elements to the dynamic arrays designated as the indicator buffers by the SetIndexBuffer() function. For indicator buffers, all size changing operations are performed by the terminal's executing subsystem.

In the source array, the elements are copied starting from the *src_start* index. The source array size remains unchanged. The elements to be added to the receiving array are not links to the source array elements. This means that subsequent changes of the elements in any of the two arrays are not reflected in the second one.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the fixed-size array and fill in the values
   int array_dest[10];
   for(int i=0;i<10;i++)
     {
      array_dest[i]=i;
     }
   //--- source array
   int array_source[10];
   for(int i=0;i<10;i++)
     {
      array_source[i]=10+i;
     }
//--- display arrays before inserting the elements
   Print("Before calling ArrayInsert()");
   ArrayPrint(array_dest);
   ArrayPrint(array_source);
//--- insert 3 elements from the source array and show the new set of the receiving a
   ArrayInsert(array_dest,array_source,4,0,3);
   Print("After calling ArrayInsert()");
   ArrayPrint(array_dest);
/*
  Execution result
   Before calling ArrayInsert()
   0 1 2 3 4 5 6 7 8 9
   After calling ArrayInsert()
   0 1 2 3 10 11 12 7 8 9
*/
```

See also

ArrayRemove, ArrayCopy, ArrayResize, ArrayFree

ArrayRemove

Removes the specified number of elements from the array starting with a specified index.

```
bool  ArrayRemove(
   void&         array[],             // array of any type
   uint          start,               // index the removal starts from
   uint          count=WHOLE_ARRAY    // number of elements
   );
```

Parameters

array[]

[in][out] Array.

start

[in] Index, starting from which the array elements are removed.

count=WHOLE_ARRAY

[in] Number of removed elements. The WHOLE_ARRAY value means removing all elements from the specified index up the end of the array.

Return Value

Returns true if successful, otherwise - false. To get information about the error, call the GetLastError() function. Possible errors:

- 5052 – ERR_SMALL_ARRAY (too big *start* value),
- 5056 – ERR_SERIES_ARRAY (the array cannot be changed, indicator buffer),
- 4003 – ERR_INVALID_PARAMETER (too big *count* value),
- 4005 - ERR_STRUCT_WITHOBJECTS_ORCLASS (fixed-size array containing complex objects with the destructor),
- 4006 - ERR_INVALID_ARRAY (fixed-size array containing structure or class objects with a destructor).

Note

If the function is used for a fixed-size array, the array size does not change: the remaining "tail" is physically copied to the *start* position. For accurate understanding of how the function works, see the example below. "Physical" copying means the copied objects are not created by calling the constructor or copying operator. Instead, the binary representation of an object is copied. For this reason, you cannot apply the ArrayRemove() function to the fixed-size array containing objects with the destructor (the ERR_INVALID_ARRAY or ERR_STRUCT_WITHOBJECTS_ORCLASS error is activated). When removing such an object, the destructor should be called twice – for the original object and its copy.

You cannot remove elements from dynamic arrays designated as the indicator buffers by the SetIndexBuffer() function. This will result in the ERR_SERIES_ARRAY error. For indicator buffers, all size changing operations are performed by the terminal's executing subsystem.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the fixed-size array and fill in the values
   int array[10];
   for(int i=0;i<10;i++)
     {
      array[i]=i;
     }
//--- display the array before removing the elements
   Print("Before calling ArrayRemove()");
   ArrayPrint(array);
//--- delete 2 elements from the array and display the new set
   ArrayRemove(array,4,2);
   Print("After calling ArrayRemove()");
   ArrayPrint(array);
/*
  Execution result:
  Before calling ArrayRemove()
   0 1 2 3 4 5 6 7 8 9
  After calling ArrayRemove()
   0 1 2 3 6 7 8 9 8 9
*/
```

See also

ArrayInsert, ArrayCopy, ArrayResize, ArrayFree

ArrayReverse

Reverses the specified number of elements in the array starting with a specified index.

```
bool  ArrayReverse(
   void&         array[],            // array of any type
   uint          start=0,            // index to start reversing the array from
   uint          count=WHOLE_ARRAY   // number of elements
   );
```

Parameters

array[]

[in][out] Array.

start=0

[in] Index the array reversal starts from.

count=WHOLE_ARRAY

[in] Number of reversed elements. If WHOLE_ARRAY, then all array elements are moved in the inversed manner starting with the specified *start* index up to the end of the array.

Return Value

Returns true if successful, otherwise - false.

Note

The ArraySetAsSeries() function does not move the array elements physically. Instead, it only changes the indexation direction backwards to arrange the access to the elements as in the timeseries. The ArrayReverse() function physically moves the array elements so that the array is "reversed".

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the fixed-size array and fill in the values
   int array[10];
   for(int i=0;i<10;i++)
     {
      array[i]=i;
     }
//--- display the array before reversing the elements
   Print("Before calling ArrayReverse()");
   ArrayPrint(array);
//--- reverse 3 elements in the array and show the new set
   ArrayReverse(array,4,3);
   Print("After calling ArrayReverse()");
   ArrayPrint(array);
/*
  Execution result:
  Before calling ArrayReverse()
   0 1 2 3 4 5 6 7 8 9
  After calling ArrayReverse()
   0 1 2 3 6 5 4 7 8 9
*/
```

See also

ArrayInsert, ArrayRemove, ArrayCopy, ArrayResize, ArrayFree, ArrayGetAsSeries, ArraySetAsSeries

ArraySetAsSeries

The function sets the AS_SERIES flag to a selected object of a dynamic array, and elements will be indexed like in timeseries.

```
bool  ArraySetAsSeries(
   const void&  array[],     // array by reference
   bool         flag         // true denotes reverse order of indexing
   );
```

Parameters

array[]

[in][out] Numeric array to set.

flag

[in] Array indexing direction.

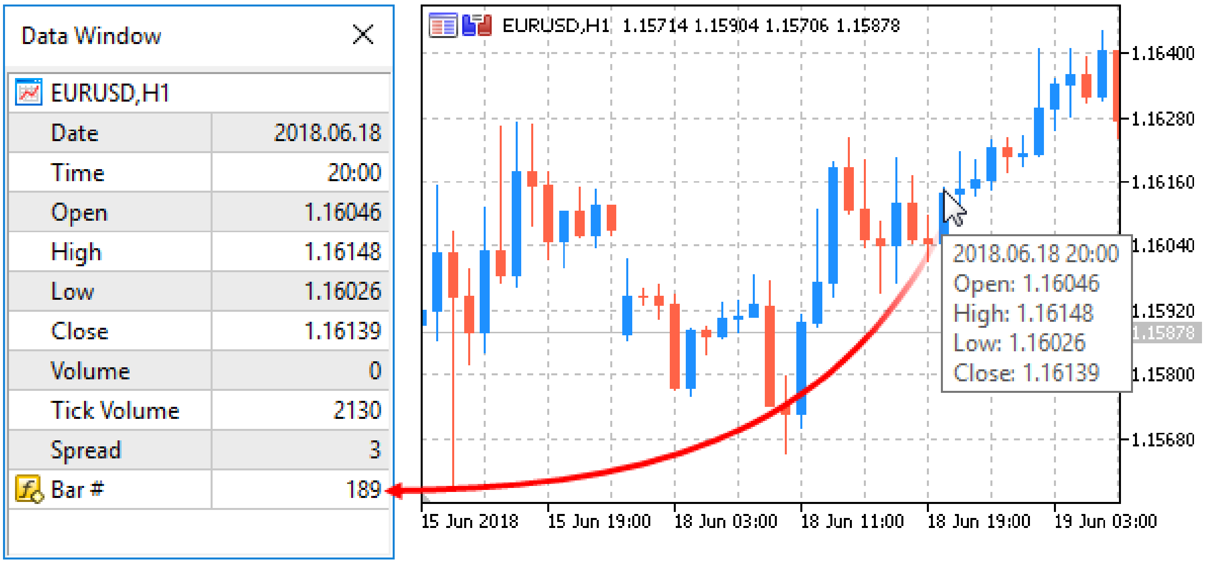
Return Value

The function returns true on success, otherwise - false.

Note

The AS_SERIES flag can't be set for multi-dimensional arrays or static arrays (arrays, whose size in square brackets is preset already on the compilation stage). Indexing in timeseries differs from a common array in that the elements of timeseries are indexed from the end towards the beginning (from the newest to oldest data).

Example: Indicator that shows bar number



Data Window

EURUSD,H1

Date	2018.06.18
Time	20:00
Open	1.16046
High	1.16148
Low	1.16026
Close	1.16139
Volume	0
Tick Volume	2130
Spread	3
Bar #	189

EURUSD,H1 1.15714 1.15904 1.15706 1.15878

2018.06.18 20:00
Open: 1.16046
High: 1.16148
Low: 1.16026
Close: 1.16139

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Numeration
```

```
#property indicator_label1  "Numeration"
#property indicator_type1   DRAW_LINE
#property indicator_color1  CLR_NONE
//--- indicator buffers
double         NumerationBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- set indexing for the buffer like in timeseries
   ArraySetAsSeries(NumerationBuffer,true);
//--- set accuracy of showing in DataWindow
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- how the name of the indicator array is displayed in DataWindow
   PlotIndexSetString(0,PLOT_LABEL,"Bar #");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---  we'll store the time of the current zero bar opening
   static datetime currentBarTimeOpen=0;
//--- revert access to array time[] - do it like in timeseries
   ArraySetAsSeries(time,true);
//--- If time of zero bar differs from the stored one
   if(currentBarTimeOpen!=time[0])
     {
     //--- enumerate all bars from the current to the chart depth
      for(int i=rates_total-1;i>=0;i--) NumerationBuffer[i]=i;
      currentBarTimeOpen=time[0];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries, ArrayGetAsSeries

ArraySize

The function returns the number of elements of a selected array.

```
int  ArraySize(
   const void&  array[]    // checked array
   );
```

Parameters

array[]

[in] Array of any type.

Return Value

Value of int type.

Note

For a one-dimensional array, the value to be returned by the ArraySize is equal to that of ArrayRange(array,0).

Example:

```
void OnStart()
  {
//--- create arrays
   double one_dim[];
   double four_dim[][10][5][2];
//--- sizes
   int one_dim_size=25;
   int reserve=20;
   int four_dim_size=5;
//--- auxiliary variable
   int size;
//--- allocate memory without backup
   ArrayResize(one_dim,one_dim_size);
   ArrayResize(four_dim,four_dim_size);
//--- 1. one-dimensional array
   Print("+==========================================================+");
   Print("Array sizes:");
   Print("1. One-dimensional array");
   size=ArraySize(one_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",one_dim_size,size);
//--- 2. multidimensional array
   Print("2. Multidimensional array");
   size=ArraySize(four_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",four_dim_size,size);
//--- dimension sizes
   int d_1=ArrayRange(four_dim,1);
   int d_2=ArrayRange(four_dim,2);
   int d_3=ArrayRange(four_dim,3);
   Print("Check:");
   Print("Zero dimension = Array size / (First dimension * Second dimension * Third di
   PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,d_3);
//--- 3. one-dimensional array with memory backup
   Print("3. One-dimensional array with memory backup");
//--- double the value
   one_dim_size*=2;
//--- allocate memory with backup
   ArrayResize(one_dim,one_dim_size,reserve);
//--- print out the size
   size=ArraySize(one_dim);
   PrintFormat("Size with backup = %d, Actual array size = %d",one_dim_size+reserve,si
  }
```

ArraySort

Sorts the values in the first dimension of a multidimensional numeric array in the ascending order.

```
bool  ArraySort(
   void&  array[]      // array for sorting
   );
```

Parameters

array[]

[in][out] Numeric array for sorting.

Return Value

The function returns true on success, otherwise - false.

Note

An array is always sorted in the ascending order irrespective of the AS_SERIES flag value.

Functions ArraySort and ArrayBSearch accept any-dimensional arrays as a parameter. However, searching and sorting are always applied to the first (zero) dimension.

Example:

```
#property description "The indicator analyzes data for the last month and draws all ca
#property description "and large tick volumes. The tick volume array is sorted out"
#property description "to define such candlesticks. The candlesticks having the volum
#property description "per cent of the array are considered small. The candlesticks ha
#property description "the last InpBigVolume per cent of the array are considered larg
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot
#property indicator_label1  "VolumeFactor"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrDodgerBlue,clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- predefined constant
#define INDICATOR_EMPTY_VALUE 0.0
//--- input parameters
input int InpSmallVolume=15; // Percentage value of small volumes (<50)
input int InpBigVolume=20;   // Percentage value of large volumes (<50)
//--- analysis start time (will be shifted)
datetime ExtStartTime;
//--- indicator buffers
double   ExtOpenBuff[];
double   ExtHighBuff[];
double   ExtLowBuff[];
double    ExtCloseBuff[];
double    ExtColorBuff[];
//--- volume boundary values for displaying the candlesticks
long      ExtLeftBorder=0;
long      ExtRightBorder=0;
//+------------------------------------------------------------------+
//| Receive border values for tick volumes                           |
//+------------------------------------------------------------------+
bool GetVolumeBorders(void)
  {
//--- variables
   datetime stop_time;  // copy end time
   long     buff[];     // buffer for copying
//--- end time is the current one
   stop_time=TimeCurrent();
//--- start time is one month earlier from the current one
   ExtStartTime=GetStartTime(stop_time);
//--- receive the values of tick volumes
   ResetLastError();
   if(CopyTickVolume(Symbol(),Period(),ExtStartTime,stop_time,buff)==-1)
     {
      //--- failed to receive the data, return false to launch recalculation command
      PrintFormat("Failed to receive tick volume values. Error code = %d",GetLastErro
      return(false);
     }
//--- calculate array size
   int size=ArraySize(buff);
//--- sort out the array
   ArraySort(buff);
//--- define the values of the left and right border for tick volumes
   ExtLeftBorder=buff[size*InpSmallVolume/100];
   ExtRightBorder=buff[(size-1)*(100-InpBigVolume)/100];
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Receive the data that is one month less than the passed one      |
//+------------------------------------------------------------------+
datetime GetStartTime(const datetime stop_time)
  {
//--- convert end time into MqlDateTime type structure variable
   MqlDateTime temp;
   TimeToStruct(stop_time,temp);
//--- receive the data that is one month less
   if(temp.mon>1)
      temp.mon-=1;  // the current month is not the first one in the year, therefore,
   else
     {
      temp.mon=12;  // the current month is the first in the year, therefore, the num
      temp.year-=1; // while the year number is one less
     }
//--- day number will not exceed 28
   if(temp.day>28)
      temp.day=28;
//--- return the obtained date
   return(StructToTime(temp));
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check if input parameters satisfy the conditions
   if(InpSmallVolume<0 || InpSmallVolume>=50 || InpBigVolume<0 || InpBigVolume>=50)
     {
      Print("Incorrect input parameters");
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtOpenBuff);
   SetIndexBuffer(1,ExtHighBuff);
   SetIndexBuffer(2,ExtLowBuff);
   SetIndexBuffer(3,ExtCloseBuff);
   SetIndexBuffer(4,ExtColorBuff,INDICATOR_COLOR_INDEX);
//--- set the value that will not be displayed
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- set labels for indicator buffers
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- check if unhandled bars are still present
   if(prev_calculated<rates_total)
     {
      //--- receive new values of the right and left borders for volumes
      if(!GetVolumeBorders())
         return(0);
     }
//--- start variable for bar calculation
   int start=prev_calculated;
//--- work at the last bar if the indicator values have already been calculated at the
   if(start>0)
      start--;
//--- set direct indexing in time series
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
   ArraySetAsSeries(tick_volume,false);
//--- the loop of calculation of the indicator values
   for(int i=start;i<rates_total;i++)
     {
      //--- fill out candlesticks starting from the initial date
      if(ExtStartTime<=time[i])
        {
         //--- if the value is not less than the right border, fill out the candlestic
         if(tick_volume[i]>=ExtRightBorder)
           {
            //--- receive data for drawing the candlestick
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- DodgerBlue color
            ExtColorBuff[i]=0;
            //--- continue the loop
            continue;
           }
         //--- fill out the candlestick if the value does not exceed the left border
         if(tick_volume[i]<=ExtLeftBorder)
           {
            //--- receive data for drawing the candlestick
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- Orange color
            ExtColorBuff[i]=1;
            //--- continue the loop
            continue;
           }
        }
      //--- set empty values for bars that have not been included in the calculation
      ExtOpenBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtHighBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtLowBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtCloseBuff[i]=INDICATOR_EMPTY_VALUE;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

ArrayBsearch

ArraySwap

Swaps the contents of two dynamic arrays of the same type. For multidimensional arrays, the number of elements in all dimensions except the first one should match.

```
bool  ArraySwap(
   void&   array1[],       // first array
   void&   array2[]        // second array
   );
```

Parameters

array1[]

[in][out] Array of numerical type.

array2[]

[in][out] Array of numerical type.

Return Value

Returns true if successful, otherwise false. In this case, GetLastError() returns the ERR_INVALID_ARRAY error code.

Note

The function accepts dynamic arrays of the same type and the same dimensions except the first one. For integer types, the sign is ignored, i.e. char==uchar)

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- arrays for storing quotes
   double source_array[][8];
   double   dest_array[][8];
   MqlRates rates[];
//--- get the data of the last 20 candles on the current timeframe
   int copied=CopyRates(NULL,0,0,20,rates);
   if(copied<=0)
     {
      PrintFormat("CopyRates(%s,0,0,20,rates) failed, error=%d",
                  Symbol(),GetLastError());
      return;
     }
//--- set the array size for the amount of copied data
   ArrayResize(source_array,copied);
//--- fill the rate_array_1[] array by data from rates[]
   for(int i=0;i<copied;i++)
     {
      source_array[i][0]=(double)rates[i].time;
      source_array[i][1]=rates[i].open;
      source_array[i][2]=rates[i].high;
      source_array[i][3]=rates[i].low;
      source_array[i][4]=rates[i].close;
      source_array[i][5]=(double)rates[i].tick_volume;
      source_array[i][6]=(double)rates[i].spread;
      source_array[i][7]=(double)rates[i].real_volume;
     }
//--- swap data between source_array[] and dest_array[]
   if(!ArraySwap(source_array,dest_array))
     {
      PrintFormat("ArraySwap(source_array,rate_array_2) failed, error code=%d",GetLast
      return;
     }
//--- ensure that the source array has become zero after the swap
   PrintFormat("ArraySwap() done: ArraySize(source_array)=%d",ArraySize(source_array))
//--- display the data of the dest_array[] destination array
   ArrayPrint(dest_array);
  }
```

See also

[ArrayCopy](), [ArrayFill](), [ArrayRange](), [ArrayIsDynamic]()

ArrayToFP16

Copies an array of type float or double into an array of type ushort with the given format.

```
bool   ArrayToFP16(
   const ushort&         dst_array[],         // copy to
   const float&          src_array[],         // copy from
   ENUM_FLOAT16_FORMAT   fmt                  // format
   );
```

Overloading for the double type

```
bool   ArrayToFP16(
   const ushort&         dst_array[],         // copy to
   const double&         src_array[],         // copy from
   ENUM_FLOAT16_FORMAT   fmt                  // format
   );
```

Parameters

dst_array[]

[out] Receiver array or type ushort.

src_array[]

[in] Source array of type float or double.

fmt

[in] Copying format from the ENUM_FLOAT16_FORMAT enumeration.

Return Value

Returns true if successful or false otherwise.

Note

Formats FLOAT16 and BFLOAT16 are defined in the ENUM_FLOAT16_FORMAT enumeration and are used in MQL5 only for operations with ONNX models.

The function converts input parameters of type float or double to type FLOAT16 and BFLOAT16. These input parameters are then used in the OnnxRun function.

FLOAT16, also known as half-precision float, uses 16 bits to represent floating-point numbers. This format provides a balance between accuracy and computational efficiency. FLOAT16 is widely used in deep learning algorithms and neural networks, which require high-performance processing of large datasets. This format accelerates computations calculations by reducing the size of numbers, which is especially important when training deep neural networks on GPUs.

BFLOAT16 (or Brain Floating Point 16) also uses 16 bits but differs from FLOAT16 in the approach to format representation. In this format, 8 bits are allocated for representing the exponent, while the remaining 7 bits are used for representing the mantissa. This format was developed for use in deep learning and artificial intelligence, especially in Google's Tensor Processing Unit (TPU). BFLOAT16 demonstrates excellent performance in neural network training and can effectively accelerate computations.

Example: function from the article Working with ONNX models in float16 and float8 formats

```
//+------------------------------------------------------------------+
//| RunCastFloat16ToDouble                                           |
//+------------------------------------------------------------------+
bool RunCastFloat16ToDouble(long model_handle)
  {
   PrintFormat("test=%s",__FUNCTION__);
   double test_data[12]= {1,2,3,4,5,6,7,8,9,10,11,12};
   ushort data_uint16[12];
   if(!ArrayToFP16(data_uint16,test_data,FLOAT_FP16))
     {
      Print("error in ArrayToFP16. error code=",GetLastError());
      return(false);
     }
   Print("test array:");
   ArrayPrint(test_data);
   Print("ArrayToFP16:");
   ArrayPrint(data_uint16);
   U<ushort> input_float16_values[3*4];
   U<double> output_double_values[3*4];
   float test_data_float[];
   if(!ArrayFromFP16(test_data_float,data_uint16,FLOAT_FP16))
     {
      Print("error in ArrayFromFP16. error code=",GetLastError());
      return(false);
     }
   for(int i=0; i<12; i++)
     {
      input_float16_values[i].value=data_uint16[i];
      PrintFormat("%d input value =%f  Hex float16 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array:");
   ArrayPrint(input_float16_values);
   bool res=OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float16_values,output_double
   if(!res)
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      return(false);
     }
   Print("ONNX output array:");
   ArrayPrint(output_double_values);
//---
   double sum_error=0.0;
   for(int i=0; i<12; i++)
```

```
     {
      double delta=test_data[i]-output_double_values[i].value;
      sum_error+=MathAbs(delta);
      PrintFormat("%d output double %f = %s  difference=%f",i,output_double_values[i].
     }
//---
   PrintFormat("test=%s   sum_error=%f",__FUNCTION__,sum_error);
//---
   return(true);
  }
```

See also

ArrayFromFP16, ArrayCopy

ArrayToFP8

Copies an array of type float or double into an array of type uchar with the given format.

```
bool   ArrayToFP8(
   const uchar&          dst_array[],         // copy to
   const float&          src_array[],         // copy from
   ENUM_FLOAT8_FORMAT    fmt                  // format
   );
```

Overloading for the double type

```
bool   ArrayToFP8(
   const uchar&          dst_array[],         // copy to
   const double&         src_array[],         // copy from
   ENUM_FLOAT8_FORMAT    fmt                  // format
   );
```

Parameters

dst_array[]

[out] Receiver array or type uchar.

src_array[]

[in] Source array of type float or double.

fmt

[in] Copying format from the ENUM_FLOAT8_FORMAT enumeration.

Return Value

Returns true if successful or false otherwise.

Note

All kinds of FP8 format are defined in the ENUM_FLOAT8_FORMAT enumeration and are used in MQL5 only for operations with ONNX models.

The function converts input parameters of type float or double into one of FP8 types. These input parameters are then used in the OnnxRun function.

FP8 (8-bit floating point) is one of the data types used to represent floating point numbers. In FP8, each number is represented by 8 data bits, typically divided into three components: sign, exponent and mantissa. This format offers a balance between accuracy and storage efficiency, making it attractive for applications that require memory and computational efficiency.

By employing compact number representation, FP8 reduces memory requirements and accelerates calculations. In addition, FP8 can be useful for implementing low-level operations such as arithmetic calculations and signal processing.

Example: function from the article Working with ONNX models in float16 and float8 formats

```
//+------------------------------------------------------------------+
//| RunCastFloat8Float                                               |
//+------------------------------------------------------------------+
bool RunCastFloat8ToFloat(long model_handle,const ENUM_FLOAT8_FORMAT fmt)
  {
   PrintFormat("TEST: %s(%s)",__FUNCTION__,EnumToString(fmt));
//---
   float test_data[15]   = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   uchar data_float8[15] = {};
   if(!ArrayToFP8(data_float8,test_data,fmt))
     {
      Print("error in ArrayToFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   U<uchar> input_float8_values[3*5];
   U<float> output_float_values[3*5];
   float    test_data_float[];
//--- convert float8 to float
   if(!ArrayFromFP8(test_data_float,data_float8,fmt))
     {
      Print("error in ArrayFromFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   for(uint i=0; i<data_float8.Size(); i++)
     {
      input_float8_values[i].value=data_float8[i];
      PrintFormat("%d input value =%f  Hex float8 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array: ",ArrayToString(input_float8_values));
//--- execute model (convert float8 to float using ONNX)
   if(!OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float8_values,output_float_value
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   Print("ONNX output array: ",ArrayToString(output_float_values));
//--- calculate error (compare ONNX and ArrayFromFP8 results)
   double sum_error=0.0;
   for(uint i=0; i<test_data.Size(); i++)
     {
      double delta=test_data_float[i]-(double)output_float_values[i].value;
      sum_error+=MathAbs(delta);
      PrintFormat("%d output float %f = %s difference=%f",i,output_float_values[i].va
     }
//---
   PrintFormat("%s(%s): sum_error=%f\n",__FUNCTION__,EnumToString(fmt),sum_error);
   return(true);
  }
```

See also

[ArrayFromFP8](), [ArrayCopy]()

ArrayFromFP16

Copies an array of type ushort into an array of float or double type with the given format.

```
bool   ArrayFromFP16(
   const float&          dst_array[],         // copy to
   const ushort&         src_array[],         // copy from
   ENUM_FLOAT16_FORMAT   fmt                  // format
   );
```

Overloading for the double type

```
bool   ArrayFromFP16(
   const double&         dst_array[],         // copy to
   const ushort&         src_array[],         // copy from
   ENUM_FLOAT16_FORMAT   fmt                  // format
   );
```

Parameters

dst_array[]

[out] Receiver array of type float or double.

src_array[]

[in] Source array of type ushort.

fmt

[in] Copying format from the ENUM_FLOAT16_FORMAT enumeration.

Return Value

Returns true if successful or false otherwise.

Note

Formats FLOAT16 and BFLOAT16 are defined in the ENUM_FLOAT16_FORMAT enumeration and are used in MQL5 only for operations with ONNX models.

If the output parameters obtained from the OnnxRun function execution are of type FLOAT16 and BFLOAT16, you can use this function to convert the result to float or double arrays.

FLOAT16, also known as half-precision float, uses 16 bits to represent floating-point numbers. This format provides a balance between accuracy and computational efficiency. FLOAT16 is widely used in deep learning algorithms and neural networks, which require high-performance processing of large datasets. This format accelerates computations calculations by reducing the size of numbers, which is especially important when training deep neural networks on GPUs.

BFLOAT16 (or Brain Floating Point 16) also uses 16 bits but differs from FLOAT16 in the approach to format representation. In this format, 8 bits are allocated for representing the exponent, while the remaining 7 bits are used for representing the mantissa. This format was developed for use in deep learning and artificial intelligence, especially in Google's Tensor Processing Unit (TPU). BFLOAT16 demonstrates excellent performance in neural network training and can effectively accelerate computations.

Example: function from the article Working with ONNX models in float16 and float8 formats

```
//+------------------------------------------------------------------+
//| RunCastFloat16ToDouble                                           |
//+------------------------------------------------------------------+
bool RunCastFloat16ToDouble(long model_handle)
  {
   PrintFormat("test=%s",__FUNCTION__);
   double test_data[12]= {1,2,3,4,5,6,7,8,9,10,11,12};
   ushort data_uint16[12];
   if(!ArrayToFP16(data_uint16,test_data,FLOAT_FP16))
     {
      Print("error in ArrayToFP16. error code=",GetLastError());
      return(false);
     }
   Print("test array:");
   ArrayPrint(test_data);
   Print("ArrayToFP16:");
   ArrayPrint(data_uint16);
   U<ushort> input_float16_values[3*4];
   U<double> output_double_values[3*4];
   float test_data_float[];
   if(!ArrayFromFP16(test_data_float,data_uint16,FLOAT_FP16))
     {
      Print("error in ArrayFromFP16. error code=",GetLastError());
      return(false);
     }
   for(int i=0; i<12; i++)
     {
      input_float16_values[i].value=data_uint16[i];
      PrintFormat("%d input value =%f  Hex float16 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array:");
   ArrayPrint(input_float16_values);
   bool res=OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float16_values,output_double
   if(!res)
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      return(false);
     }
   Print("ONNX output array:");
   ArrayPrint(output_double_values);
//---
   double sum_error=0.0;
   for(int i=0; i<12; i++)
```

```
     {
      double delta=test_data[i]-output_double_values[i].value;
      sum_error+=MathAbs(delta);
      PrintFormat("%d output double %f = %s  difference=%f",i,output_double_values[i].
     }
//---
   PrintFormat("test=%s   sum_error=%f",__FUNCTION__,sum_error);
//---
   return(true);
  }
```

See also

ArrayToFP16, ArrayCopy

ArrayFromFP8

Copies an array of type uchar into an array of float or double type with the given format.

```
bool   ArrayFromFP8(
   const float&          dst_array[],        // copy to
   const uchar&          src_array[],        // copy from
   ENUM_FLOAT8_FORMAT    fmt                 // format
   );
```

Overloading for the double type

```
bool   ArrayFromFP8(
   const double&         dst_array[],        // copy to
   const uchar&          src_array[],        // copy from
   ENUM_FLOAT8_FORMAT    fmt                 // format
   );
```

Parameters

dst_array[]

[out] Receiver array of type float or double.

src_array[]

[in] Source array of type uchar.

fmt

[in] Copying format from the ENUM_FLOAT8_FORMAT enumeration.

Return Value

Returns true if successful or false otherwise.

Note

All kinds of FP8 format are defined in the ENUM_FLOAT8_FORMAT enumeration and are used in MQL5 only for operations with ONNX models.

If the output parameters obtained from the OnnxRun function execution are of FP8 from the ENUM_FLOAT8_FORMAT enumeration, you can use this function to convert the result into float or double arrays.

FP8 (8-bit floating point) is one of the data types used to represent floating point numbers. In FP8, each number is represented by 8 data bits, typically divided into three components: sign, exponent and mantissa. This format offers a balance between accuracy and storage efficiency, making it attractive for applications that require memory and computational efficiency.

By employing compact number representation, FP8 reduces memory requirements and accelerates calculations. In addition, FP8 can be useful for implementing low-level operations such as arithmetic calculations and signal processing.

Example: function from the article Working with ONNX models in float16 and float8 formats

```
//+------------------------------------------------------------------+
//| RunCastFloat8Float                                               |
//+------------------------------------------------------------------+
bool RunCastFloat8ToFloat(long model_handle,const ENUM_FLOAT8_FORMAT fmt)
  {
   PrintFormat("TEST: %s(%s)",__FUNCTION__,EnumToString(fmt));
//---
   float test_data[15]   = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   uchar data_float8[15] = {};
   if(!ArrayToFP8(data_float8,test_data,fmt))
     {
      Print("error in ArrayToFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   U<uchar> input_float8_values[3*5];
   U<float> output_float_values[3*5];
   float    test_data_float[];
//--- convert float8 to float
   if(!ArrayFromFP8(test_data_float,data_float8,fmt))
     {
      Print("error in ArrayFromFP8. error code=",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   for(uint i=0; i<data_float8.Size(); i++)
     {
      input_float8_values[i].value=data_float8[i];
      PrintFormat("%d input value =%f  Hex float8 = %s  ushort value=%d",i,test_data_
     }
   Print("ONNX input array: ",ArrayToString(input_float8_values));
//--- execute model (convert float8 to float using ONNX)
   if(!OnnxRun(model_handle,ONNX_NO_CONVERSION,input_float8_values,output_float_values
     {
      PrintFormat("error in OnnxRun. error code=%d",GetLastError());
      OnnxRelease(model_handle);
      return(false);
     }
   Print("ONNX output array: ",ArrayToString(output_float_values));
//--- calculate error (compare ONNX and ArrayFromFP8 results)
   double sum_error=0.0;
   for(uint i=0; i<test_data.Size(); i++)
     {
      double delta=test_data_float[i]-(double)output_float_values[i].value;
      sum_error+=MathAbs(delta);
```

```
        PrintFormat("%d output float %f = %s difference=%f",i,output_float_values[i].val
       }
//---
   PrintFormat("%s(%s): sum_error=%f\n",__FUNCTION__,EnumToString(fmt),sum_error);
   return(true);
  }
```

See also

[ArrayToFP8](), [ArrayCopy]()

Matrices and vectors

A matrix is a two-dimensional array of double, float, or complex numbers.

A vector is a one-dimensional array of double, float, or complex numbers. The vector has no indication of whether it is vertical or horizontal. It is determined from the use context. For example, the vector operation Dot assumes that the left vector is horizontal and the right one is vertical. If the type indication is required, one-row or one-column matrices can be used. However, this is generally not necessary.

Matrices and vectors allocate memory for data dynamically. In fact, matrices and vectors are objects that have certain properties, such as the type of data they contain and dimensions. Matrix and vector properties can be obtained using methods such as vector_a.Size(), matrix_b.Rows(), vector_c.Norm(), matrix_d.Cond() and others. Any dimension can be changed.

When creating and initializing matrices, so-called static methods are used (these are like static methods of a class). For example: matrix::Eye(), matrix::Identity(), matrix::Ones(), vector::Ones(), matrix: :Zeros(), vector::Zeros(), matrix::Full(), vector::Full(), matrix::Tri().

At the moment, matrix and vector operations do not imply the use of the complex data type, as this development direction has not yet been completed.

MQL5 supports passing of matrices and vectors to DLLs. This enables the import of functions utilizing the relevant types, from external variables.

Matrices and vectors are passed to a DLL as a pointer to a buffer. For example, to pass a matrix of type float, the corresponding parameter of the function exported from the DLL must take a float-type buffer pointer.

MQL5

```
#import "mmlib.dll"
bool sgemm(uint flags, matrix<float> &C, const matrix<float> &A, const matrix<float> 
#import
```

C++

```
extern "C" __declspec(dllexport) bool sgemm(UINT flags, float *C, const float *A, cons
```

In addition to buffers, you should pass matrix and vector sizes for correct processing.

All matrix and vector methods are listed below in alphabetical order.

Function	Action	Category
Activation	Compute activation function values and write them to the passed vector/matrix	Machine learning
ArgMax	Return the index of the maximum value	Statistics
ArgMin	Return the index of the minimum value	Statistics

Function	Action	Category
ArgSort	Return the sorted index	Manipulations
Assign	Copies a matrix, vector or array with auto cast	Initialization
Average	Compute the weighted average of matrix/vector values	Statistics
Cholesky	Compute the Cholesky decomposition	Transformations
Clip	Limits the elements of a matrix/vector to a given range of valid values	Manipulations
Col	Return a column vector. Write a vector to the specified column.	Manipulations
Cols	Return the number of columns in a matrix	Features
Compare	Compare the elements of two matrices/vectors with the specified precision	Manipulations
CompareByDigits	Compare the elements of two matrices/vectors with the significant figures precision	Manipulations
Cond	Compute the condition number of a matrix	Features
Convolve	Return the discrete, linear convolution of two vectors	Products
Copy	Return a copy of the given matrix/vector	Manipulations
Concat	Concatenate 2 submatrices to one matrix. Concatenate 2 vectors to one vector	Manipulations
CopyIndicatorBuffer	Get the data of the specified indicator buffer in the specified quantity to a vector	Initialization
CopyRates	Gets the historical series of the MqlRates structure of the specified symbol-period in the specified amount into a matrix or vector	Initialization
CopyTicks	Get ticks from an MqlTick structure into a matrix or a	Initialization

Function	Action	Category
	vector	
CopyTicksRange	Get ticks from an MqlTick structure into a matrix or a vector within the specified date range	Initialization
CorrCoef	Compute the Pearson correlation coefficient (linear correlation coefficient)	Products
Correlate	Compute the cross-correlation of two vectors	Products
Cov	Compute the covariance matrix	Products
CumProd	Return the cumulative product of matrix/vector elements, including those along the given axis	Statistics
CumSum	Return the cumulative sum of matrix/vector elements, including those along the given axis	Statistics
Derivative	Compute activation function derivative values and write them to the passed vector/matrix	Machine learning
Det	Compute the determinant of a square invertible matrix	Features
Diag	Extract a diagonal or construct a diagonal matrix	Manipulations
Dot	Dot product of two vectors	Products
Eig	Computes the eigenvalues and right eigenvectors of a square matrix	Transformations
EigVals	Computes the eigenvalues of a general matrix	Transformations
Eye	Return a matrix with ones on the diagonal and zeros elsewhere	Initialization
Fill	Fill an existing matrix or vector with the specified value	Initialization

Function	Action	Category
Flat	Access a matrix element through one index instead of two	Manipulations
Full	Create and return a new matrix filled with the given value	Initialization
GeMM	The GeMM (General Matrix Multiply) method implements the general multiplication of two matrices	Products
HasNan	Return the number of NaN values in a matrix/vector	Manipulations
Hsplit	Split a matrix horizontally into multiple submatrices. Same as Split with axis=0	Manipulations
Identity	Create an identity matrix of the specified size	Initialization
Init	Matrix or vector initialization	Initialization
Inner	Inner product of two matrices	Products
Inv	Compute the multiplicative inverse of a square invertible matrix by the Jordan-Gauss method	Solutions
Kron	Return Kronecker product of two matrices, matrix and vector, vector and matrix or two vectors	Products
Loss	Compute loss function values and write them to the passed vector/matrix	Machine learning
LstSq	Return the least-squares solution of linear algebraic equations (for non-square or degenerate matrices)	Solutions
LU	Implement an LU decomposition of a matrix: the product of a lower triangular matrix and an upper triangular matrix	Transformations
LUP	Implement an LUP factorization with partial permutation, which refers to LU decomposition with row permutations only: PA=LU	Transformations

Function	Action	Category
MatMul	Matrix product of two matrices	Products
Max	Return the maximum value in a matrix/vector	Statistics
Mean	Compute the arithmetic mean of element values	Statistics
Median	Compute the median of the matrix/vector elements	Statistics
Min	Return the minimum value in a matrix/vector	Statistics
Norm	Return matrix or vector norm	Features
Ones	Create and return a new matrix filled with ones	Initialization
Outer	Compute the outer product of two matrices or two vectors	Products
Percentile	Return the specified percentile of values of matrix/vector elements or elements along the specified axis	Statistics
PInv	Compute the pseudo-inverse of a matrix by the Moore-Penrose method	Solutions
Power	Raise a square matrix to an integer power	Products
Prod	Return the product of matrix/vector elements, which can also be executed for the given axis	Statistics
Ptp	Return the range of values of a matrix/vector or of the given matrix axis	Statistics
QR	Compute the qr factorization of a matrix	Transformations
Quantile	Return the specified quantile of values of matrix/vector elements or elements along the specified axis	Statistics
Random	Static function. Create and return a new matrix or vector filled with random values.	Initialization

Function	Action	Category
	Random values are generated uniformly within the specified range	
Rank	Return matrix rank using the Gaussian method	Features
RegressionMetric	Compute the regression metric as the deviation error from the regression line constructed on the specified data array	Statistics
Reshape	Change the shape of a matrix without changing its data	Manipulations
Resize	Return a new matrix with a changed shape and size	Manipulations
Row	Return a row vector. Write the vector to the specified row	Manipulations
Rows	Return the number of rows in a matrix	Features
Set	Sets the value for a vector element by the specified index	Manipulations
Size	Return the size of vector	Features
SLogDet	Compute the sign and logarithm of the determinant of an matrix	Features
Solve	Solve a linear matrix equation or a system of linear algebraic equations	Solutions
Sort	Sort by place	Manipulations
Spectrum	Compute spectrum of a matrix as the set of its eigenvalues from the product AT*A	Features
Split	Split a matrix into multiple submatrices	Manipulations
Std	Return the standard deviation of values of matrix/vector elements or elements along the specified axis	Statistics
Sum	Return the sum of matrix/vector elements, which can also be executed for the given axis (axes)	Statistics

Function	Action	Category
SVD	Singular value decomposition	Transformations
SwapCols	Swap columns in a matrix	Manipulations
SwapRows	Swap rows in a matrix	Manipulations
Trace	Return the sum along diagonals of the matrix	Features
Transpose	Transpose (swap the axes) and return the modified matrix	Manipulations
Tri	Construct a matrix with ones on a specified diagonal and below, and zeros elsewhere	Initialization
TriL	Return a copy of a matrix with elements above the k-th diagonal zeroed. Lower triangular matrix	Manipulations
TriU	Return a copy of a matrix with the elements below the k-th diagonal zeroed. Upper triangular matrix	Manipulations
Var	Compute the variance of values of matrix/vector elements	Statistics
Vsplit	Split a matrix vertically into multiple submatrices. Same as Split with axis=1	Manipulations
Zeros	Create and return a new matrix filled with zeros	Initialization

Matrix and Vector Types

Matrix and vector are special data types in MQL5 which enable linear algebra operations. The following data types exist:

- matrix — a matrix containing double elements.
- matrixf — a matrix containing float elements.
- matrixc — a matrix containing complex elements.
- vector — a vector containing double elements.
- vectorf — a vector containing float elements.
- vectorc — a vector containing complex elements.

Template functions support notations like matrix<double>, matrix<float>, vector<double>, vector<float> instead of the corresponding types.

Matrix and vector initialization methods

Function	Action
Eye	Return a matrix with ones on the diagonal and zeros elsewhere
Identity	Create an identity matrix of the specified size
Ones	Create and return a new matrix filled with ones
Zeros	Create and return a new matrix filled with zeros
Full	Create and return a new matrix filled with given value
Tri	Construct a matrix with ones at and below the given diagonal and zeros elsewhere
Init	Initialize a matrix or a vector
Fill	Fill an existing matrix or vector with the specified value

Enumeration for matrix and vector operations

This section describes the enumerations that are used in various matrix and vector methods.

ENUM_AVERAGE_MODE

Smoothing type enumeration.

ID	Description
AVERAGE_NONE	No averaging. Results are provided for each label separately
AVERAGE_BINARY	Label 1 result for binary classification
AVERAGE_MICRO	Average error matrix result (confusion matrix)
AVERAGE_MACRO	Average result from the results of the error matrices of each label
AVERAGE_WEIGHTED	Weighted average result

ENUM_VECTOR_NORM

Enumeration of vector norms for vector::Norm.

ID	Description
VECTOR_NORM_INF	Inf norm
VECTOR_NORM_MINUS_INF	Minus Inf norm
VECTOR_NORM_P	Norm P

ENUM_MATRIX_NORM

Enumeration of matrix norms for matrix::Norm and for obtaining the matrix::Cond matrix condition number.

ID	Description
MATRIX_NORM_FROBENIUS	Frobenius norm
MATRIX_NORM_SPECTRAL	Spectral norm
MATRIX_NORM_NUCLEAR	Nuclear norm
MATRIX_NORM_INF	Inf norm
MATRIX_NORM_P1	P1 norm

ID	Description
MATRIX_NORM_P2	P2 norm
MATRIX_NORM_MINUS_INF	Minus Inf norm
MATRIX_NORM_MINUS_P1	Minus P1 norm
MATRIX_NORM_MINUS_P2	Minus P2 norm

ENUM_VECTOR_CONVOLVE

Enumeration for convolution vector::Convolve and cross-correlation vector::Correlate.

ID	Description
VECTOR_CONVOLVE_FULL	Convolve full
VECTOR_CONVOLVE_SAME	Convolve same
VECTOR_CONVOLVE_VALID	Convolve valid

ENUM_REGRESSION_METRIC

Enumeration of regression metrics for vector::RegressionMetric.

ID	Description
REGRESSION_MAE	Mean Absolute Error
REGRESSION_MSE	Mean Squared Error
REGRESSION_RMSE	Root Mean Squared Error
REGRESSION_R2	R-Squared
REGRESSION_MAPE	Mean Absolute Percentage Error
REGRESSION_MSPE	Mean Squared Percentage Error
REGRESSION_RMSLE	Root Mean Squared Logarithmic Error
REGRESSION_SMAPE	Symmetric Mean Absolute Percentage Error
REGRESSION_MAXE	Maximal Absolute Error
REGRESSION_MEDE	Median Absolute Error
REGRESSION_MPD	Mean Poisson Deviance
REGRESSION_MGD	Mean Gamma Deviance
REGRESSION_EXPV	Explained Variance

ENUM_CLASSIFICATION_METRIC

Enumeration of metrics for classification problems.

ID	Description
CLASSIFICATION_ACCURACY	Model quality in terms of prediction accuracy for all classes
CLASSIFICATION_AVERAGE_PRECISION	Average model accuracy
CLASSIFICATION_BALANCED_ACCURACY	Balanced prediction accuracy
CLASSIFICATION_F1	F1 score. Harmonic mean between the model precision and recall
CLASSIFICATION_JACCARD	Jaccard score
CLASSIFICATION_PRECISION	Model accuracy in predicting true positives for the target class
CLASSIFICATION_RECALL	Model completeness
CLASSIFICATION_ROC_AUC	Area under the error curve
CLASSIFICATION_TOP_K_ACCURACY	Frequency of the correct label appearing at the top of k predicted labels

ENUM_LOSS_FUNCTION

Enumeration for loss function calculations vector::Loss.

ID	Description
LOSS_MSE	Root mean square error
LOSS_MAE	Mean Absolute Error
LOSS_CCE	Categorical Crossentropy
LOSS_BCE	Binary Crossentropy
LOSS_MAPE	Mean Absolute Percentage Error
LOSS_MSLE	Mean Squared Logarithmic Error
LOSS_KLD	Kullback-Leibler Divergence
LOSS_COSINE	Cosine similarity/proximity
LOSS_POISSON	Poisson
LOSS_HINGE	Hinge

ID	Description
LOSS_SQ_HINGE	Squared Hinge
LOSS_CAT_HINGE	Categorical Hinge
LOSS_LOG_COSH	Logarithm of the Hyperbolic Cosine
LOSS_HUBER	Huber

ENUM_ACTIVATION_FUNCTION

Enumeration for the activation function vector::Activation and for the activation function derivative vector::Derivative.

ID	Description
AF_ELU	Exponential Linear Unit
AF_EXP	Exponential
AF_GELU	Gaussian Error Linear Unit
AF_HARD_SIGMOID	Hard Sigmoid
AF_LINEAR	Linear
AF_LRELU	Leaky Rectified Linear Unit
AF_RELU	REctified Linear Unit
AF_SELU	Scaled Exponential Linear Unit
AF_SIGMOID	Sigmoid
AF_SOFTMAX	Softmax
AF_SOFTPLUS	Softplus
AF_SOFTSIGN	Softsign
AF_SWISH	Swish
AF_TANH	The hyperbolic tangent function
AF_TRELU	Thresholded Rectified Linear Unit

ENUM_SORT_MODE

Enumeration of sort types for the Sort function.

ID	Description
SORT_ASCENDING	Sort ascending

ID	Description
SORT_DESCENDING	Sort descending

ENUM_MATRIX_AXIS

Enumeration for specifying the axis in all statistical functions for matrices.

ID	Description
AXIS_NONE	The axis is not specified. Calculation is performed over all matrix elements, as if it were a vector (see the Flat method).
AXIS_HORZ	Horizontal axis
AXIS_VERT	Vertical axis

Initialization

There are several ways to declare and initialize matrices and vectors.

Function	Action
Assign	Copies a matrix, vector or array with auto cast
CopyIndicatorBuffer	Get the data of the specified indicator buffer in the specified quantity to a vector
CopyRates	Gets the historical series of the MqlRates structure of the specified symbol-period in the specified amount into a matrix or vector
CopyTicks	Get ticks from an MqlTick structure into a matrix or a vector
CopyTicksRange	Get ticks from an MqlTick structure into a matrix or a vector within the specified date range
Eye	Return a matrix with ones on the diagonal and zeros elsewhere
Identity	Create an identity matrix of the specified size
Ones	Create and return a new matrix filled with ones
Zeros	Create and return a new matrix filled with zeros
Full	Create and return a new matrix filled with given value
Tri	Construct a matrix with ones at and below the given diagonal and zeros elsewhere
Init	Initialize a matrix or a vector
Fill	Fill an existing matrix or vector with the specified value
Random	Static function. Create and return a new matrix or vector filled with random values. Random values are generated uniformly within the specified range

Declaration without specifying the size (no memory allocation for the data):

```
matrix          matrix_a;   // double type matrix
matrix<double> matrix_a1;  // another way to declare a double matrix; can be used i
matrixf         matrix_a2;  // float matrix
matrix<float>   matrix_a3;  // float matrix
vector          vector_a;   // double vector
vector<double> vector_a1;
vectorf         vector_a2;  // float vector
vector<float>   vector_a3;
```

Declaration with the specified size (with memory allocation for the data, but without any initialization):

```
matrix          matrix_a(128,128);           // the parameters can be either constant
```

```
  matrix<double> matrix_a1(InpRows,InpCols);  // or variables
  matrixf        matrix_a2(1,128);            // analog of a horizontal vector
  matrix<float>  matrix_a3(InpRows,1);        // analog of a vertical vector
  vector         vector_a(256);
  vector<double> vector_a1(InpSize);
  vectorf        vector_a2(SomeFunc());       // function SomeFunc returns a number o
  vector<float>  vector_a3(InpSize+16);       // expression can be used as a paramete
```

Declaration with initialization (matrix and vector sizes are determined by the initializing sequence):

```
  matrix         matrix_a={{0.1,0.2,0.3},{0.4,0.5,0.6}};
  matrix<double> matrix_a1=matrix_a;                      // must be matrices of the
  matrixf        matrix_a2={{1,0,0},{0,1,0},{0,0,1}};
  matrix<float>  matrix_a3={{1,2},{3,4}};
  vector         vector_a={-5,-4,-3,-2,-1,0,1,2,3,4,5};
  vector<double> vector_a1={1,5,2.4,3.3};
  vectorf        vector_a2={0,1,2,3};
  vector<float>  vector_a3=vector_a2;                     // must be vectors of the s
```

Declaration with initialization:

```
template<typename T>
void MatrixArange(matrix<T> &mat,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<mat.Rows(); i++)
     {
      for(ulong j=0; j<mat.Cols(); j++,value+=step)
         mat[i][j]=value;
     }
  }
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
...

  matrix  matrix_a(size_m,size_k,MatrixArange,-M_PI,0.1); // first an uninitialized ma
  matrixf matrix_a1(10,20,MatrixArange);                  // after creating the matri
  vector  vector_a(size,VectorArange,-10.0);              // after creating the vecto
  vectorf vector_a1(128,VectorArange);
```

Please note that the matrix or vector dimensions can be changed, since the memory for data is always dynamic.

Static methods

Static methods for creating matrices and vectors of the specified size, initialized in a certain way:

```
matrix         matrix_a =matrix::Eye(4,5,1);
matrix<double> matrix_a1=matrix::Full(3,4,M_PI);
matrixf        matrix_a2=matrixf::Identity(5,5);
matrixf<float> matrix_a3=matrixf::Ones(5,5);
matrix         matrix_a4=matrix::Tri(4,5,-1);
vector         vector_a =vector::Ones(256);
vectorf        vector_a1=vector<float>::Zeros(16);
vector<float>  vector_a2=vectorf::Full(128,float_value);
```

Methods for initializing already created matrices and vectors:

```
matrix  matrix_a;
matrix_a.Init(size_m,size_k,MatrixArange,-M_PI,0.1);
matrixf matrix_a1(3,4);
matrix_a1.Init(10,20,MatrixArange);
vector  vector_a;
vector_a.Init(128,VectorArange);
vectorf vector_a1(10);
vector_a1.Init(vector_size,VectorArange,start_value,step);

matrix_a.Fill(double_value);
vector_a1.Fill(FLT_MIN);
matrix_a1.Identity();
```

Assign

Copies a matrix, vector or array with auto cast.

```
bool matrix::Assign(
  const matrix<T>   &mat      // copied matrix
   );
bool matrix::Assign(
  const void        &array[] // copied array
   );
bool vector::Assign(
  const vector<T>   &vec      // copied vector
   );
bool vector::Assign(
  const void        &array[] // copied array
   );
```

Parameters

m, v or array

[in] The matrix, vector or array the values are copied from.

Return Value

Returns true if successful, otherwise – false.

Note

Unlike Copy, the Assign method allows copying arrays as well. In this case, the auto cast takes place, while the resulting matrix or vector adjusts to the size of the copied array.

Example:

```
//--- copy the matrices
  matrix a= {{2, 2}, {3, 3}, {4, 4}};
  matrix b=a+2;
  matrix c;
  Print("matrix a \n", a);
  Print("matrix b \n", b);
  c.Assign(b);
  Print("matrix c \n", a);

//--- copy the array to the matrix
  matrix double_matrix=matrix::Full(2,10,3.14);
  Print("double_matrix before Assign() \n", double_matrix);
  int int_arr[5][5]= {{1, 2}, {3, 4}, {5, 6}};
  Print("int_arr: ");
  ArrayPrint(int_arr);
  double_matrix.Assign(int_arr);
```

```
   Print("double_matrix after Assign(int_arr) \n", double_matrix);
   /*
    matrix a
    [[2,2]
     [3,3]
     [4,4]]
    matrix b
    [[4,4]
     [5,5]
     [6,6]]
    matrix c
    [[2,2]
     [3,3]
     [4,4]]

    double_matrix before Assign()
    [[3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14]
     [3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14,3.14]]

    int_arr:
        [,0][,1][,2][,3][,4]
    [0,]   1   2   0   0   0
    [1,]   3   4   0   0   0
    [2,]   5   6   0   0   0
    [3,]   0   0   0   0   0
    [4,]   0   0   0   0   0

    double_matrix after Assign(int_arr)
    [[1,2,0,0,0]
     [3,4,0,0,0]
     [5,6,0,0,0]
     [0,0,0,0,0]
     [0,0,0,0,0]]

   */
```

See also

[Copy]

CopyIndicatorBuffer

Get the data of the specified indicator buffer in the specified quantity to a vector.

The data will be copied into the vector locating the oldest element at the beginning of the physical memory allocated for the vector. There are three function options.

Access by the initial position and the number of required elements

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // indicator handle
   ulong     buffer_index,         // indicator buffer number
   ulong     start_pos,            // starting position to copy
   ulong     count                 // number of elements to copy
   );
```

Access by the start date and the number of required elements

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // indicator handle
   ulong     buffer_index,         // indicator buffer number
   datetime  start_time,           // from which date to copy
   ulong     count                 // number of elements to copy
   );
```

Access by the initial and final dates of the required time interval

```
bool  vector::CopyIndicatorBuffer(
   long      indicator_handle,     // indicator handle
   ulong     buffer_index,         // indicator buffer number
   datetime  start_time,           // from which date to copy
   datetime  stop_time             // up to which date to copy
   );
```

Parameters

indicator_handle

[in] The indicator handle obtained by the relevant indicator function.

buffer_index

[in] The number of the indicator buffer.

start_pos

[in] The number of the first copied element index.

count

[in] The number of copied elements.

start_time

[in] Bar time corresponding to the first element.

stop_time

[in] Bar time corresponding to the last element.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The elements of the copied data (the indicator buffer with index buffer_index) are counted down from the present to the past, and thus the starting position equal to 0 means the current bar (the indicator value for the current bar).

When copying an unknown amount of data, you should declare a vector without specifying a size (without allocating memory for the data), since the CopyBuffer() function tries to distribute the size of the receiving vector to the size of the copied data.

When partial copying of the indicator values is needed, you should use an intermediate vector into which the required quantity is copied. From this intermediate vector, you can copy the required number of values member by member, to the required places of the receiving vector.

If you are copying a predetermined amount of data, it is recommended to pre-declare a vector and specify its size to avoid unnecessary memory reallocation.

When requesting data from an indicator, the function immediately returns false if requested timeseries have not yet been constructed or they need to be downloaded from the server, while it initiates loading/constructing.

When requesting data from an EA or a script, download from the server is initiated if the terminal does not have the appropriate data locally, or construction of the necessary timeseries starts if the data can be constructed from the local history but the required timeframes are not ready yet. The function returns the amount that will be ready by the time the timeout expires.

See also

CopyBuffer

CopyRates

Gets the historical series of the MqlRates structure of the specified symbol-period in the specified amount into a matrix or vector. Elements are counted down from the present to the past, which means that the starting position equal to 0 means the current bar.

The data is copied so that the oldest element is placed at the beginning of the matrix/vector. There are three function options.

Access by the initial position and the number of required elements

```
bool  matrix::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   ulong            start,         // index of the initial bar copying starts from
   ulong            count          // how many to copy
   );
```

Access by the initial date and the number of required elements

```
bool  matrix::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   datetime         from,          // start date
   ulong            count          // how many to copy
   );
```

Access by the initial and final dates of the required time interval

```
bool  matrix::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   datetime         from,          // start date
   datetime         to             // end date
   );
```

Vector Methods

Access by the initial position and the number of required elements

```
bool  vector::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   ulong            start,         // index of the initial bar copying starts from
   ulong            count          // how many to copy
   );
```

Access by the initial date and the number of required elements

```
bool  vector::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   datetime         from,          // start date
   ulong            count          // how many to copy
   );
```

Access by the initial and final dates of the required time interval

```
bool  vector::CopyRates(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   ulong            rates_mask,    // combination of flags to specify the requested se
   datetime         from,          // start date
   datetime         to             // end date
   );
```

Parameters

symbol

[in] Symbol.

period

[in] Period.

rates_mask

[in] The ENUM_COPY_RATES enumeration combination of flags specifying the type of requested series. When copying to a vector, only one value from the ENUM_COPY_RATES enumeration can be specified, otherwise an error occurs.

start

[in] First copied element index.

count

[in] Number of copied elements.

from

[in] Bar time corresponding to the first element.

to

[in] Bar time corresponding to the last element.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If the interval of the requested data is completely outside the available data on the server, then the function returns false. If the data outside TERMINAL_MAXBARS (maximum number of bars on the chart) is requested, the function also returns false.

When requesting data from an EA or a script, download from the server is initiated if the terminal does not have the appropriate data locally, or construction of the necessary timeseries starts if the data can be constructed from the local history but they are not ready yet. The function returns the amount that will be ready by the time the timeout expires, however the history download continues, and the function returns more data during the next similar request.

When requesting data by the start date and the number of required items, only data whose date is less than (before) or equal to the specified one is returned. The interval is set and considered up to a second. In other words, the opening date of any bar the value is returned for (volume, spread, Open, High, Low, Close or Time) is always equal to or less than the specified one.

When requesting data in a given date range, only data that falls within the requested interval is returned. The interval is set and considered up to a second. In other words, the opening time of any bar the value is returned for (volume, spread, indicator buffer value, Open, High, Low, Close or Time) is always located in the requested interval.

For example, if the current day of the week is Saturday, the function returns 0 when attempting to copy data on the weekly timeframe by setting *start_time*=*Last_Tuesday* and *stop_time*=*Last_Friday* because the open time on the weekly timeframe always falls on Sunday, but not a single weekly bar falls within the specified range.

If you need to get the value corresponding to the current incomplete bar, then you can use the first call form indicating *start_pos*=0 and *count*=1.

ENUM_COPY_RATES

The ENUM_COPY_RATES enumeration contains the flags to specify the type of data to be passed to the matrix or array. The flag combination allows getting several series from the history in one request. The order of the rows in the matrix will correspond to the order of the values in the ENUM_COPY_RATES enumeration. In other words, the row with High data will always be higher in the matrix than the row with Low data.

ID	Value	Description
COPY_RATES_OPEN	1	Open price series
COPY_RATES_HIGH	2	High price series
COPY_RATES_LOW	4	Low price series
COPY_RATES_CLOSE	8	Close price series
COPY_RATES_TIME	16	Time series (bar open time) Getting time in float of the vector and the matrix (vectord and matrixf) causes losses of ~100 seconds since float accuracy is severely limited and integers greater than 1<<24 cannot be accurately represented in float.
COPY_RATES_VOLUME_TICK	32	Tick Volumes
COPY_RATES_VOLUME_REAL	64	Trade Volumes

ID	Value	Description
COPY_RATES_SPREAD	128	Spreads
Combination		
COPY_RATES_OHLC	15	Open, High, Low and Close series
COPY_RATES_OHLCT	31	Open, High, Low, Close and Time series
Data arrangement		
COPY_RATES_VERTICAL	32768	Series are copied into the matrix along the vertical axis. The received series values will be arranged vertically in the matrix, i.e., the oldest data will be in the first row, while the most recent data will be in the last matrix row. With default copying, series are added into a matrix along the horizontal axis. The flag is only applicable when copying to a matrix.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- get quotes to the matrix
  matrix matrix_rates;
  if(matrix_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_OHLCT, 1, 10))
    Print("matrix rates: \n", matrix_rates);
  else
    Print("matrix_rates.CopyRates failed. Error ", GetLastError());
//--- check
  MqlRates mql_rates[];
  if(CopyRates(Symbol(), PERIOD_CURRENT, 1, 10, mql_rates)>0)
   {
    Print("mql_rates array:");
    ArrayPrint(mql_rates);
   }
  else
    Print("CopyRates(Symbol(), PERIOD_CURRENT,1, 10, mql_rates). Error ", GetLastErro
//--- get quotes into the vector = invalid call
  vector vector_rates;
  if(vector_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_OHLC, 1, 15))
    Print("vector_rates COPY_RATES_OHLC: \n", vector_rates);
  else
```

```
    Print("vector_rates.CopyRates COPY_RATES_OHLC failed. Error ", GetLastError());
//--- get close prices into the vector
  if(vector_rates.CopyRates(Symbol(), PERIOD_CURRENT, COPY_RATES_CLOSE, 1, 15))
    Print("vector_rates COPY_RATES_CLOSE: \n", vector_rates);
  else
    Print("vector_rates.CopyRates failed. Error ", GetLastError());
 };
/*
   matrix rates:
   [[0.99686,0.99638,0.99588,0.99441,0.99464,0.99594,0.99698,0.99758,0.99581,0.995280
    [0.99708,0.99643,0.99591,0.9955000000000001,0.99652,0.99795,0.99865,0.99764,0.996
    [0.9961100000000001,0.99491,0.99426,0.99441,0.99448,0.99494,0.9964499999999999,0.
    [0.99641,0.99588,0.99441,0.99464,0.99594,0.99697,0.99758,0.99581,0.99528000000000
    [1662436800,1662440400,1662444000,1662447600,1662451200,1662454800,1662458400,166
   mql_rates array:
                    [time]   [open]   [high]    [low] [close] [tick_volume] [spread] [rea
   [0] 2022.09.06 04:00:00 0.99686 0.99708 0.99611 0.99641          4463        0
   [1] 2022.09.06 05:00:00 0.99638 0.99643 0.99491 0.99588          4519        0
   [2] 2022.09.06 06:00:00 0.99588 0.99591 0.99426 0.99441          3060        0
   [3] 2022.09.06 07:00:00 0.99441 0.99550 0.99441 0.99464          3867        0
   [4] 2022.09.06 08:00:00 0.99464 0.99652 0.99448 0.99594          5280        0
   [5] 2022.09.06 09:00:00 0.99594 0.99795 0.99494 0.99697          7227        0
   [6] 2022.09.06 10:00:00 0.99698 0.99865 0.99645 0.99758         10130        0
   [7] 2022.09.06 11:00:00 0.99758 0.99764 0.99472 0.99581          7012        0
   [8] 2022.09.06 12:00:00 0.99581 0.99604 0.99360 0.99528          6166        0
   [9] 2022.09.06 13:00:00 0.99528 0.99570 0.99220 0.99259          6950        0
   vector_rates.CopyRates COPY_RATES_OHLC failed. Error 4003
   vector_rates COPY_RATES_CLOSE:
   [0.9931,0.99293,0.99417,0.99504,0.9968399999999999,0.99641,0.99588,0.99441,0.99464,
*/
```

See also

[Access to Timeseries and Indicators](), [CopyRates]()

CopyTicks

Get ticks from an MqlTick structure into a matrix or a vector. Elements are counted from the past to the present, which means that the tick with index 0 is the oldest one. To analyze a tick, check the *flags* field which shows what exactly has changed in the tick.

```
bool  matrix::CopyTicks(
   string           symbol,                 // symbol name
   ulong            flags,                  // flag indicating the type of ticks to be
   ulong            from_msc,               // time from which ticks are requested
   ulong            count                   // number of ticks to be received
   );
```

Vector Method

```
bool  vector::CopyTicks(
   string           symbol,                 // symbol name
   ulong            flags,                  // flag indicating the type of ticks to be
   ulong            from_msc,               // time from which ticks are requested
   ulong            count                   // number of ticks to be received
   );
```

Parameters

symbol

[in] Symbol.

flags

[in] A combination of flags from the ENUM_COPY_TICKS enumeration indicating the contents of the requested data. When copying to a vector, you can specify only one value from the ENUM_COPY_TICKS enumeration, otherwise an error will occur.

from_msc

[in] Time starting from which ticks are requested. Time is specified in milliseconds since 01/01/1970. If from_msc=0, the last number of ticks equal to 'count' are returned.

count

[in] The number of requested ticks. If parameters 'from_msc' and 'count' are not specified, all available ticks, but no more than 2000, will be written.

Return Value

Returns true on success or false if error occurs.

Note

The first call of CopyTicks() initiates synchronization of the relevant symbol's tick database stored on the hard drive. If the local database does not provide all the requested ticks, then missing ticks will be automatically downloaded from the trade server. Ticks from ***from_msc*** specified in CopyTicks() to the current moment will be synchronized. After that, all ticks arriving for this symbol will be added to the tick database thus keeping it in the synchronized state.

If *from_msc* and *count* parameters are not specified, all available ticks, but no more than 2000, will be written to the matrix/vector.

In indicators, the CopyTicks() method returns the result immediately: When called from an indicator, CopyTick() immediately returns all available ticks of a symbol and launches synchronization of the tick database if available data is not enough. All indicators on the same symbol operate in one common thread, so the indicator cannot wait for the completion of synchronization. After synchronization, CopyTicks() will return all requested ticks during the next call. In indicators, the OnCalculate() function is called after the arrival of each tick.

In Expert Advisors and scripts, CopyTicks() can wait for the result for 45 seconds: as distinct from indicators, every Expert Advisor or script operates in a separate thread, and therefore can wait up to 45 seconds for the synchronization to complete. If the required amount of ticks fails to be synchronized during this time, CopyTicks() will return available ticks by timeout and will continue synchronization. OnTick() in Expert Advisors is not a handler of every tick, while it only notifies the Expert Advisor about changes in the market. This can be a batch of changes: the terminal can simultaneously receive multiple ticks, while OnTick() will be called only once, to notify the Expert Advisor about the latest market state.

Rate of data return: the terminal stores 4096 last ticks for each instrument in the fast access cache (65536 ticks for symbols with the Market Depth running). Requests concerning this data are executed the fastest. If requested ticks for the current trading session are beyond the cache, CopyTicks() calls the ticks stored in the terminal memory. These requests require more time to complete. The slowest requests are those requesting ticks for other days, since the data is read from the drive in this case.

ENUM_COPY_TICKS

The ENUM_COPY_TICKS enumeration contains the flags to specify the type of data to be passed to the matrix or array. The flag combination allows getting several series from the history in one request. The order of the rows in the matrix will correspond to the order of the values in the ENUM_COPY_TICKS enumeration. In other words, the row with High data will always be higher in the matrix than the row with Low data.

ID	Value	Description
COPY_TICKS_INFO	1	All ticks
COPY_TICKS_TRADE	2	Ticks containing Bid and/or Ask price changes
COPY_TICKS_ALL	3	Ticks containing Last and/or Volume price changes
COPY_TICKS_TIME_MS	1<<8	Tick time in milliseconds
COPY_TICKS_BID	1<<9	Bid price
COPY_TICKS_ASK	1<<10	Ask price
COPY_TICKS_LAST	1<<11	Last price (last deal price)
COPY_TICKS_VOLUME	1<<12	Last price Volume

ID	Value	Description
COPY_TICKS_FLAGS	1<<13	Tick Flags
Data arrangement		
COPY_TICKS_VERTICAL	1<<15	Ticks are copied into the matrix along the vertical axis. The received ticks will be arranged vertically in the matrix, i.e., the oldest ticks will be in the first row, while the most recent ticks will be in the last matrix row. With default copying, ticks are added into a matrix along the horizontal axis. The flag is only applicable when copying to a matrix.

Analyze the tick flags to find out which data has changed:

- TICK_FLAG_BID – the tick has changed the bid price
- TICK_FLAG_ASK – the tick has changed the ask price
- TICK_FLAG_LAST – the tick has changed the last deal price
- TICK_FLAG_VOLUME – the tick has changed the volume
- TICK_FLAG_BUY – the tick is a result of a buy deal
- TICK_FLAG_SELL – the tick is a result of a sell deal

See also

Access to Timeseries and Indicators, CopyTicks

CopyTicksRange

Get ticks from an MqlTick structure into a matrix or a vector within the specified date range. Elements are counted from the past to the present, which means that the tick with index 0 is the oldest one. To analyze a tick, check the *flags* field which shows what exactly has changed in the tick.

```
bool  matrix::CopyTicksRange(
   string           symbol,                // symbol name
   ulong            flags,                 // flag indicating the type of ticks to be
   ulong            from_msc,              // time from which ticks are requested
   ulong            to_msc                 // time up to which ticks are requested
   );
```

Vector Method

```
bool  vector::CopyTicksRange(
   string           symbol,                // symbol name
   ulong            flags,                 // flag indicating the type of ticks to be
   ulong            from_msc,              // time from which ticks are requested
   ulong            to_msc                 // time up to which ticks are requested
   );
```

Parameters

symbol

[in] Symbol.

flags

[in] A combination of flags from the ENUM_COPY_TICKS enumeration indicating the contents of the requested data. When copying to a vector, you can specify only one value from the ENUM_COPY_TICKS enumeration, otherwise an error will occur.

from_msc

[in] Time starting from which ticks are requested. Time is specified in milliseconds since 01/01/1970. If the 'from_msc' parameter is not specified, ticks from the beginning of the history are returned. Ticks with the time >= from_msc will be returned.

to_msc

[in] Time up to which ticks are requested. Time is specified in milliseconds since 01/01/1970. Ticks with the time <= to_msc are returned. If the to_msc parameter is not specified, all ticks up to the history end are returned.

Return Value

Returns true on success or false if error occurs. GetLastError() can return the following errors:

- ERR_HISTORY_TIMEOUT – timeout for tick synchronization has expired, the function has returned all it had.
- ERR_HISTORY_SMALL_BUFFER – static buffer is too small. Only the amount the array can store has been returned.
- ERR_NOT_ENOUGH_MEMORY – not enough memory to receive historical data from the specified range into a dynamic tick array. Failed to allocate enough memory for the tick array.

Analyze the tick flags to find out which data has changed:

- TICK_FLAG_BID – the tick has changed the bid price
- TICK_FLAG_ASK – the tick has changed the ask price
- TICK_FLAG_LAST – the tick has changed the last deal price
- TICK_FLAG_VOLUME – the tick has changed the volume
- TICK_FLAG_BUY – the tick is a result of a buy deal
- TICK_FLAG_SELL – the tick is a result of a sell deal

Note

The CopyTicksRange() method is used to request ticks from exactly the specified range. For example, ticks for a specific day in history. CopyTicks() allows specifying only the start date, for example, to receive all ticks from the beginning of the month up to now.

See also

Access to Timeseries and Indicators, CopyTicksRange

Eye

A static function. Constructs a matrix having a specified size with ones on the main diagonal and zeros elsewhere. Returns a matrix with ones on the diagonal and zeros elsewhere.

```
static matrix matrix::Eye(
  const ulong  rows,        // number of rows
  const ulong  cols,        // number of columns
  const int    ndiag=0      // index of the diagonal
   );
```

Parameters

rows

[in] Number of rows in the output.

cols

[in] Number of columns in the output.

ndiag=0

[in] Index of the diagonal: 0 (the default) refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal.

Return Value

A matrix where all elements are equal to zero, except for the k-th diagonal, whose values are equal to one.

MQL5 example:

```
  matrix eye=matrix::Eye(3, 3);
  Print("eye = \n", eye);

  eye=matrix::Eye(4, 4,1);
  Print("eye = \n", eye);
  /*
   eye =
   [[1,0,0]
    [0,1,0]
    [0,0,1]]
   eye =
   [[0,1,0,0]
    [0,0,1,0]
    [0,0,0,1]
    [0,0,0,0]]
  */
```

Python example:

```
np.eye(3, dtype=int)
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])


 np.eye(4, k=1)
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

Identity

It is a static function which creates an identity matrix of the specified size (not necessarily square). An identity matrix contains ones on the main diagonal and zeros elsewhere. The main diagonal consists of the matrix elements having equal row and column indexes, such as [0,0],[1,1],[2,2] etc. Creates a new identity matrix.

There is also the method Identity that transforms an already existing matrix into an identity one.

```
static matrix matrix::Identity(
  const ulong  rows,        // number of rows
  const ulong  cols,        // number of columns
   );

void matrix::Identity();
```

Parameters

rows

[in] Number of rows (and columns) in n x n matrix.

Return Value

Return the identity matrix. The identity matrix is a square matrix with ones on the main diagonal.

MQL5 example:

```
  matrix identity=matrix::Identity(3,3);
  Print("identity = \n", identity);
/*
   identity =
   [[1,0,0]
    [0,1,0]
    [0,0,1]]
*/
  matrix identity2(3,5);
  identity2.Identity();
  Print("identity2 = \n", identity2);
/*
   identity2 =
   [[1,0,0,0,0]
    [0,1,0,0,0]
    [0,0,1,0,0]]
*/
```

Python example:

```
np.identity(3)
array([[1.,  0.,  0.],
        [0.,   1.,   0.],
        [0.,   0.,   1.]])
```

Ones

This is a static function that creates and returns a new matrix filled with ones.

```
static matrix matrix::Ones(
  const ulong  rows,      // number of rows
  const ulong  cols       // number of columns
   );

static vector vector::Ones(
  const ulong  size,      // vector size
   );
```

Parameters

rows

[in] Number of rows.

cols

[in] Number of columns.

Return Value

A new matrix of given rows and columns, filled with ones.

MQL5 example:

```
  matrix ones=matrix::Ones(4, 4);
  Print("ones = \n", ones);
/*
ones =
   [[1,1,1,1]
    [1,1,1,1]
    [1,1,1,1]
    [1,1,1,1]]
*/
```

Python example:

```
np.ones((4, 1))
array([[1.],
       [1.]])
```

Zeros

This is a static function that creates and returns a new matrix filled with zeros.

```
static matrix matrix::Zeros(
  const ulong  rows,      // number of rows
  const ulong  cols       // number of columns
   );

static vector vector::Zeros(
  const ulong  size,      // vector size
   );
```

Parameters

rows

[in] Number of rows.

cols

[in] Number of columns.

Return Value

A new matrix of given rows and columns, filled with zeros.

MQL5 example:

```
  matrix zeros=matrix::Zeros(3, 4);
  Print("zeros = \n", zeros);
/*
zeros =
   [[0,0,0,0]
    [0,0,0,0]
    [0,0,0,0]]
*/
```

Python example:

```
np.zeros((2, 1))
array([[ 0.],
       [ 0.]])
```

Full

The static function creates and returns a new matrix filled with given value.

```
static matrix matrix::Full(
  const ulong   rows,       // number or rows
  const ulong   cols,       // number of columns
  const double  value       // value to fill
   );

static vector vector::Full(
  const ulong   size,       // vector size
  const double  value       // value to fill
   );
```

Parameters

rows

[in] Number of rows.

cols

[in] Number of columns.

value

[in] Value to fill all the matrix elements.

Return Value

Return a new matrix of given rows and columns, filled with specified value.

MQL5 example:

```
  matrix full=matrix::Full(3,4,10);
  Print("full = \n", full);
/*
full =
   [[10,10,10,10]
    [10,10,10,10]
    [10,10,10,10]]
*/
```

Example

```
np.full((2, 2), 10)
array([[10, 10],
       [10, 10]])
```

Tri

This is a static function Construct a matrix with ones at and below the given diagonal and zeros elsewhere.

```
static matrix matrix::Tri(
  const ulong rows,         // number of rows
  const ulong cols,         // number of columns
  const int   ndiag=0       // number of diagonal
   );
```

Parameters

rows

[in] Number of rows in the array.

cols

[in] Number of columns in the array.

ndiag=0

[in] The sub-diagonal at and below which the array is filled. k = 0 is the main diagonal, while k < 0 is below it, and k > 0 is above. The default is 0.

Return Value

Array with its lower triangle filled with ones and zero elsewhere.

MQL5 example:

```
   matrix matrix_a=matrix::Tri(3,4,1);
   Print("Tri(3,4,1)\n",matrix_a);
   matrix_a=matrix::Tri(4,3,-1);
   Print("Tri(4,3,-1)\n",matrix_a);

/*
   Tri(3,4,1)
   [[1,1,0,0]
    [1,1,1,0]
    [1,1,1,1]]
   Tri(4,3,-1)
   [[0,0,0]
    [1,0,0]
    [1,1,0]
    [1,1,1]]
*/
```

Example

```
np.tri(3, 5, 2, dtype=int)
array([[1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0],
       [1, 1, 1, 1, 1]])
```

Init

Initialize a matrix or a vector.

```
void matrix::Init(
  const ulong  rows,             // number of rows
  const ulong  cols,             // number of columns
  func_name    init_func=NULL,   // init function placed in some scope or static method
   ...         parameters
   );

void vector::Init(
  const ulong  size,             // vector size
  func_name    init_func=NULL,   // init function placed in some scope or static method
   ...         parameters
   );
```

Parameters

rows

[in] Number of rows.

cols

[in] Number of columns.

func_name

[in] Initializing function.

...

[in] Parameters of the initializing function.

Return Value

No return value.

Example

```
template<typename T>
void MatrixArange(matrix<T> &mat,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<mat.Rows(); i++)
     {
      for(ulong j=0; j<mat.Cols(); j++,value+=step)
         mat[i][j]=value;
     }
  }
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
  int size_m=3, size_k=4;
  matrix  m(size_m,size_k,MatrixArange,-2.,0.1); // first an uninitialized matrix size
  Print("matrix m \n",m);                        // then function MatrixArange with th
  matrixf m_float(5,5,MatrixArange,-2.f,0.1f);   // after a matrix of float type is cr
  Print("matrix m_float \n",m_float);
  vector  v(size_k,VectorArange,-10.0);          // after a vector is created, VectorA
  Print("vector v \n",v);
  /*
   matrix m
   [[-2,-1.9,-1.8,-1.7]
    [-1.6,-1.5,-1.399999999999999,-1.299999999999999]
    [-1.199999999999999,-1.099999999999999,-0.9999999999999992,-0.8999999999999992]]
   matrix m_float
   [[-2,-1.9,-1.8,-1.6999999,-1.5999999]
    [-1.4999999,-1.3999999,-1.2999998,-1.1999998,-1.0999998]
    [-0.99999976,-0.89999974,-0.79999971,-0.69999969,-0.59999967]
    [-0.49999967,-0.39999968,-0.29999968,-0.19999969,-0.099999689]
    [3.1292439e-07,0.10000031,0.20000032,0.30000031,0.4000003]]
   vector v
   [-10,-9,-8,-7]
  */
  }
```

Fill

Fill an existing matrix or vector with the specified value.

```
void matrix::Fill(
  const double  value       // value to fill
   );

void vector::Fill(
  const double  value       // value to fill
   );
```

Parameters

value

[in] Value to fill all the matrix elements.

Return Value

No return value. The matrix is filled in place with the specified value.

Example

```
   matrix matrix_a(2,2);
   matrix_a.Fill(10);
   Print("matrix_a\n",matrix_a);

/*
  matrix_a
  [[10,10]
   [10,10]]
*/
```

Random

Static function. Create and return a new matrix or vector filled with random values. Random values are generated uniformly within the specified range.

```
static vector vector::Random(
  const ulong   size,        // vector length
  const double  min=0.0,     // minimum value
  const double  max=1.0      // maximum value
   );

static matrix matrix::Random(
  const ulong   rows,        // number of rows
  const ulong   cols         // number of columns
  const float   min=0.0,     // minimum value
  const float   max=1.0      // maximum value
   );
```

The function fills an existing matrix or vector with random values. Random values are generated uniformly within the specified range.

```
void  vector::Random(
  const double  min=0.0,     // minimum value
  const double  max=1.0      // maximum value
   );

void  matrix::Random(
  const float  min=0.0,      // minimum value
  const float  max=1.0       // maximum value
   );
```

Parameters

rows

[in] Number of rows.

cols

[in] Number of columns.

size

[in] Vector length.

min=0.0

[in] The minimum value in the generated sample of random numbers.

max=1.0

[in] The maximum value in the generated sample of random numbers.

Return Value

Return a new matrix of given rows and columns filled with random values.

Example

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector v1= vector::Random(6,0,4);
   Print("vector v1 \n", v1);
   /*
   vector v1
   [3.340834286841758,0.9073984578501895,3.15598511117417,0.7106475841045956,2.2940100
   */

   vector v2(4);
   v2.Random(-3,3);
   Print("vector v2 \n", v2);
   /*
   vector v2
   [-2.642032080246609,1.008521607147651,1.340623848219547,-2.789016363161853]
   */

   matrix m1 = matrix::Random(5, 5, -1, 3);
   Print("matrix m1 \n", m1);
   /*
   matrix m1
   [[1.468433369525809,1.199383023707052,0.6154667729547869,2.830012157519816,-0.06995
    [2.638591175814763,2.966658948912904,2.001837257994767,-0.08570802469870731,0.8260
    [0.982665140643405,0.5825661973987162,-0.6615881071569371,0.4069533109215444,-0.16
    [-0.5791836119208447,-0.9332020478167575,2.801898707589577,-0.09887318478405316,-0
    [-0.1069569017521471,-0.04576879345014895,0.6320100840255642,-0.3805729270401813,0
   */

   matrix m2(3, 3);
   m2.Random(-2, 2);
   Print("matrix m2 \n", m2);
   /*
   matrix m2
   [[-1.928191291151613,-1.222873445419332,-1.965333905062949]
    [-1.077723600686929,1.053317863273741,-0.1786824246353196]
    [-1.965241401448204,1.620956906029246,-1.696386492740453]]
   */
  }
```

Matrix and vector manipulations

These are methods for basic matrix operations: filling, copying, getting a part of a matrix, transposing, splitting and sorting.

There are also several methods for operations with matrix rows and columns.

Function	Action
HasNan	Return the number of NaN values in a matrix/vector
Transpose	Reverse or permute the axes of a matrix; returns the modified matrix
TransposeConjugate	Transposing a complex matrix with conjugation. Reverse or permute the axes of a matrix by changing a sign of an imaginary part of a complex number, return the modified matrix
TriL	Return a copy of a matrix with elements above the k-th diagonal zeroed. Lower triangular matrix
TriU	Return a copy of a matrix with elements below the k-th diagonal zeroed. Upper triangular matrix
Diag	Extract a diagonal or construct a diagonal matrix
Row	Return a row vector. Write a vector to the specified row
Col	Return a column vector. Write a vector to the specified column
Copy	Return a copy of the given matrix/vector
Concat	Concatenate 2 submatrices to one matrix. Concatenate 2 vectors to one vector
Compare	Compare the elements of two matrices/vectors with the specified precision
CompareByDigits	Compare the elements of two matrices/vectors up to significant digits
CompareEqual	Perform an absolute comparison of two matrices by unfolding successive rows into one-dimensional vectors
Flat	Allows addressing a matrix element through one index instead of two
Clip	Limit the elements of a matrix/vector to a specified range of valid values
Reshape	Change the shape of a matrix without changing its data
Resize	Return a new matrix with a changed shape and size
SwapRows	Swap rows in a matrix
SwapCols	Swap columns in a matrix
Split	Split a matrix into multiple submatrices

Function	Action
Hsplit	Split a matrix horizontally into multiple submatrices. Same as Split with axis=0
Vsplit	Split a matrix vertically into multiple submatrices. Same as Split with axis=1
ArgSort	Indirectly sort a matrix or vector.
Sort	Sort a matrix or vector in place.

HasNan

Return the number of NaN values in a matrix/vector.

```
ulong vector::HasNan();

ulong matrix::HasNan();
```

Return Value

The number of matrix/vector elements that contain a NaN value.

Note

When comparing the appropriate pair of elements having NaN values, the Compare and CompareByDigits methods consider these elements equal, while in case of a usual comparison of floating-point numbers NaN != NaN.

Example:

```
void OnStart(void)
  {
   double x=sqrt(-1);

   Print("single: ",x==x);

   vector<double> v1={x};
   vector<double> v2={x};

   Print("vector: ", v1.Compare(v2,0)==0);
  }

/* Result:

 single: false
 vector: true
*/
```

See also

MathClassify, Compare, CompareByDigits

Transpose

Matrix transposition. Reverse or permute the axes of a matrix; returns the modified matrix.

```
matrix matrix::Transpose()
```

Return Value

Transposed matrix.

A simple matrix transposition algorithm in MQL5:

```
matrix MatrixTranspose(const matrix& matrix_a)
  {
   matrix matrix_c(matrix_a.Cols(),matrix_a.Rows());

   for(ulong i=0; i<matrix_c.Rows(); i++)
      for(ulong j=0; j<matrix_c.Cols(); j++)
         matrix_c[i][j]=matrix_a[j][i];

   return(matrix_c);
  }
```

MQL5 example:

```
  matrix a= {{0, 1, 2}, {3, 4, 5}};
  Print("matrix a \n", a);
  Print("a.Transpose() \n", a.Transpose());

  /*
  matrix a
  [[0,1,2]
   [3,4,5]]
  a.Transpose()
  [[0,3]
   [1,4]
   [2,5]]
  */
```

Python example:

```
import numpy as np

a = np.arange(6).reshape((2,3))
print("a \n",a)
print("np.transpose(a) \n",np.transpose(a))
```

```
a
 [[0 1 2]
 [3 4 5]]
np.transpose(a)
 [[0 3]
 [1 4]
 [2 5]]
```

TransposeConjugate

Transposing a complex matrix with conjugation. Reverse or permute the axes of a matrix by changing a sign of an imaginary part of a complex number, return the modified matrix.

```
matrix matrix::TransposeConjugate()
```

Return Value

Transposed complex conjugate matrix.

Simple algorithm of transposing a complex matrix with conjugation - method explanation

```
//--- Complex matrix transpose function with conjugation
matrix MatrixTranspose(const matrix& matrix_a) {
    //--- create a new matrix_c with dimensions inverse to matrix_a
    matrix matrix_c(matrix_a.Cols(), matrix_a.Rows());

    //--- go through all the rows of the new matrix
    for (ulong i = 0; i < matrix_c.Rows(); i++) {
        //--- go through all the columns of the new matrix
        for (ulong j = 0; j < matrix_c.Cols(); j++) {
            //--- transfer the real part of the element by transposing the index
            matrix_c[i][j].real = matrix_a[j][i].real;

            //--- transfer the imaginary part of the element by changing the sign (co
            matrix_c[i][j].imag = -matrix_a[j][i].imag;
        }
    }

    //--- return the transposed matrix with conjugation
    return (matrix_c);
}
```

TriL

Return a copy of a matrix with elements above the k-th diagonal zeroed. Lower triangular matrix.

```
matrix matrix::Tril(
  const int      ndiag=0      // index of diagonal
   );
```

Parameters

ndiag=0

[in] Diagonal above which to zero elements. ndiag = 0 (the default) is the main diagonal, ndiag < 0 is below it and ndiag > 0 is above.

Return Value

Array with its lower triangle filled with ones and zero elsewhere.

MQL5 example:

```
   matrix a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   matrix b=a.TriL(-1);
   Print("matrix b \n",b);

  /*
  matrix_c
  [[0,0,0]
  [4,0,0]
  [7,8,0]
  [10,11,12]]
  */
```

Python example:

```
import numpy as np

a=np.tril([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)

[[ 0  0  0]
 [ 4  0  0]
 [ 7  8  0]
 [10 11 12]]
```

TriU

Return a copy of a matrix with the elements below the k-th diagonal zeroed. Upper triangular matrix.

```
matrix matrix::Triu(
  const int      ndiag=0      // index of diagonal
   );
```

Parameters

ndiag=0

[in] Diagonal below which to zero elements. ndiag = 0 (the default) is the main diagonal, ndiag < 0 is below it and ndiag > 0 is above.

MQL5 example:

```
   matrix a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   matrix b=a.TriU(-1);
   Print("matrix b \n",b);

  /*
  matrix b
  [[1,2,3]
   [4,5,6]
   [0,8,9]
   [0,0,12]]
  */
```

Python example:

```
import numpy as np

a=np.triu([[1,2,3],[4,5,6],[7,8,9],[10,11,12]], -1)
print(a)

[[ 1  2  3]
 [ 4  5  6]
 [ 0  8  9]
 [ 0  0 12]]
```

Diag

Extract a diagonal or construct a diagonal matrix.

```
vector matrix::Diag(
  const int      ndiag=0       // number of diagonal
   );

void matrix::Diag(
  const vector  v,             // diagonal vector
  const int      ndiag=0       // number of diagonal
   );
```

Parameters

v

[in] A vector whose elements will be contained in the corresponding diagonal (ndiag=0 is the main diagonal).

ndiag=0

[in] Diagonal in question. The default is 0. Use ndiag>0 for diagonals above the main diagonal, and ndiag<0 for diagonals below the main diagonal.

Note

A diagonal can be set for unallocated matrices (which do not have dimensions). In this case, a zero matrix of the size will be created with the size corresponding to the size of the diagonal vector, after which the vector values will be populated in the corresponding diagonal. If the diagonal is set to an already existing matrix, the matrix dimensions do not change and the values of the matrix elements outside the diagonal vector do not change.

Example

```
   vector v1={1,2,3};
   matrix m1;
   m1.Diag(v1);
   Print("m1\n",m1);
   matrix m2;
   m2.Diag(v1,-1);
   Print("m2\n",m2);
   matrix m3;
   m3.Diag(v1,1);
   Print("m3\n",m3);
   matrix m4=matrix::Full(4,5,9);
   m4.Diag(v1,1);
   Print("m4\n",m4);

   Print("diag -1 - ",m4.Diag(-1));
```

```
  Print("diag 0 - ",m4.Diag());
  Print("diag 1 - ",m4.Diag(1));

 /*

 m1
 [[1,0,0]
 [0,2,0]
 [0,0,3]]
 m2
 [[0,0,0]
 [1,0,0]
 [0,2,0]
 [0,0,3]]
 m3
 [[0,1,0,0]
 [0,0,2,0]
 [0,0,0,3]]
 m4
 [[9,1,9,9,9]
 [9,9,2,9,9]
 [9,9,9,3,9]
 [9,9,9,9,9]]
 diag -1 - [9,9,9]
 diag 0 - [9,9,9,9]
 diag 1 - [1,2,3,9]
 */
```

Row

Return a row vector. Write a vector to the specified row

```
vector matrix::Row(
  const ulong   nrow       // row number
   );

void matrix::Row(
  const vector  v,         // row vector
  const ulong   nrow       // row number
   );
```

Parameters

nrow

[in] Number of row.

Return Value

Vector.

Note

A row can be set for unallocated matrices (which do not have dimensions). In this case, a zero matrix will be created with the size of the vector size x row number+1, after which the values of the vector elements will be populated in the corresponding row. If the row is set to an already existing matrix, the matrix dimensions do not change and the values of the matrix elements outside the row vector do not change.

Example

```
   vector v1={1,2,3};
   matrix m1;
   m1.Row(v1,1);
   Print("m1\n",m1);
   matrix m2=matrix::Full(4,5,7);
   m2.Row(v1,2);
   Print("m2\n",m2);

   Print("row 1 - ",m2.Row(1));
   Print("row 2 - ",m2.Row(2));

  /*
  m1
  [[0,0,0]
  [1,2,3]]
  m2
```

```
[[7,7,7,7,7]
 [7,7,7,7,7]
 [1,2,3,7,7]
 [7,7,7,7,7]]
 row 1 - [7,7,7,7,7]
 row 2 - [1,2,3,7,7]
 */
```

Col

Return a column vector. Write a vector to the specified column.

```
vector matrix::Col(
  const ulong   ncol       // column number
   );

void matrix::Col(
  const vector  v,         // column vector
  const ulong   ncol       // column number
   );
```

Parameters

ncol

[in] Number of column.

Return Value

Vector.

Note

A column can be set for unallocated matrices (which do not have dimensions). In this case, a zero matrix will be created with the size of the vector size x column number+1, after which the values of the vector elements will be populated in the corresponding column. If the column is set to an already existing matrix, the matrix dimensions do not change and the values of the matrix elements outside the column vector do not change.

Example

```
   vector v1={1,2,3};
   matrix m1;
   m1.Col(v1,1);
   Print("m1\n",m1);
   matrix m2=matrix::Full(4,5,8);
   m2.Col(v1,2);
   Print("m2\n",m2);

   Print("col 1 - ",m2.Col(1));
   Print("col 2 - ",m2.Col(2));

  /*
  m1
  [[0,1]
  [0,2]
  [0,3]]
  m2
```

```
[[8,8,1,8,8]
 [8,8,2,8,8]
 [8,8,3,8,8]
 [8,8,8,8,8]]
col 1 - [8,8,8,8]
col 2 - [1,2,3,8]
*/
```

Copy

Create a copy of the given matrix/vector.

```
bool matrix::Copy(
  const matrix&  a      // copied matrix
   );
bool vector::Copy(
  const vector&  v      // copied vector
   );
```

Parameters

v

[in] Matrix or vector to copy.

Return Value

Returns true on success, false otherwise.

MQL5 example:

```
  matrix a=matrix::Eye(3, 4);
  matrix b;
  b.Copy(a);
  matrix c=a;
  Print("matrix b \n", b);
  Print("matrix_c \n", c);

  /*
  /*
  matrix b
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]]
  matrix_c
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]]
  */
  */
```

Python example:

```
import numpy as np

a = np.eye(3,4)
print('a \n',a)
b = a
```

```
print('b \n',b)
c = np.copy(a)
print('c \n',c)

a
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
b
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
c
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

Concat

Concatenate 2 submatrices to one matrix. Concatenate 2 vectors to one vector.

```
vector vector::Concat(
  const vector&   second_part        // second vector to concatenate
   );

vector matrix::Concat(
  const matrix&   second_part        // second matrix to concatenate
   );

matrix matrix::Split(
  const matrix&   second_part,       // second matrix to concatenate
  const int       axis               // axis
   );
```

Parameters

second_part

[in] Second vector or matrix to concatenate. If matrices are concatenated along any of the axes, then the sizes of the matrices must be consistent according to the axis.

axis

[in] Axis. 0 - horizontal axis, 1 - vertical axis.

Return Value

Returns vector if vectors or matrices concatenated without axis parameter, or matrix concatenated along horizontal or vertical axis.

Example

```
   vector vector_a={1,2,3,4};
   vector vector_b={5,6,7};
   vector vector_c=vector_a.Concat(vector_b);
   Print("vector_c=",vector_c);

   matrix matrix_a={{1,2,3},{4,5,6}};
   matrix matrix_b={{7,8,9},{10,11,12}};
   vector_c=matrix_a.Concat(matrix_b);
   Print("vector_c=",vector_c);

   matrix matrix_c0=matrix_a.Concat(matrix_b,0);
   Print("matrix_c0=\n",matrix_c0);

   matrix matrix_c1=matrix_a.Concat(matrix_b,1);
   Print("matrix_c1=\n",matrix_c1);

  /*
```

```
vector_c=[1,2,3,4,5,6,7]
vector_c=[1,2,3,4,5,6,7,8,9,10,11,12]
matrix_c0=
[[1,2,3]
 [4,5,6]
 [7,8,9]
 [10,11,12]]
matrix_c1=
[[1,2,3,7,8,9]
 [4,5,6,10,11,12]]
*/
```

Compare

Compare the elements of two matrices/vectors with the specified precision.

```
ulong vector::Compare(
  const vector& vec,          // vector to compare
  const double  epsilon        // precision
   );

ulong matrix::Compare(
  const matrix& mat,           // matrix to compare
  const double  epsilon        // precision
   );
```

Parameters

vec

[in] Vector to compare.

mat

[in] Matrix to compare.

epsilon

[in] Precision.

Return Value

The number of mismatched elements of the matrices or vectors being compared: 0 if the matrices are equal, greater than 0 otherwise.

Note

The comparison operators == or != execute an exact element-wise comparison. It is known that the exact comparison of real numbers is of limited use, so the epsilon comparison method was added. It may happen that one matrix can contain elements in a range, for example from 1e-20 to 1e+20. Such matrices can be processed using element-wise comparison up to significant figures.

For complex matrices/vectors, the comparison involves estimating the distance between complex numbers. The distance is calculated as sqrt(pow(r1-r2, 2) + pow(i1-i2, 2) and is a real number that can already be compared with epsilon.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4}};
   matrix matrix_i=matrix::Identity(3,3);
   matrix matrix_c=matrix_a.Inv();
   matrix matrix_check=matrix_a.MatMul(matrix_c);
   Print("matrix_check\n",matrix_check);

   ulong errors=matrix_check.Compare(matrix::Identity(3,3),1e-15);
```

```
 Print("errors=",errors);


/*
matrix_check
[[1,0,0]
[4.440892098500626e-16,1,8.881784197001252e-16]
[4.440892098500626e-16,2.220446049250313e-16,0.9999999999999996]]
errors=0

*/
```

CompareByDigits

Compare the elements of two matrices/vectors with the significant digits precision.

```
ulong vector::CompareByDigits(
  const vector& vec,          // vector to compare
  const int     digits        // number of significant digits
   );

ulong matrix::CompareByDigits(
  const matrix& mat,          // matrix to compare
  const int     digits        // number of significant digits
   );
```

Parameters

vec

[in] Vector to compare.

mat

[in] Matrix to compare.

digits

[in] Number of significant digits to compare.

Return Value

The number of mismatched elements of the matrices or vectors being compared: 0 if the matrices are equal, greater than 0 otherwise.

Note

The comparison operators == or != execute an exact element-wise comparison. It is known that the exact comparison of real numbers is of limited use, so the epsilon comparison method was added. It may happen that one matrix can contain elements in a range, for example from 1e-20 to 1e+20. Such matrices can be processed using element-wise comparison up to significant digits.

Example

```
   int    size_m=128;
   int    size_k=256;
   matrix matrix_a(size_m,size_k);
//--- fill the matrix
   double  value=0.0;
   for(int i=0; i<size_m; i++)
     {
      for(int j=0; j<size_k; j++)
        {
         if(i==j)
```

```
             matrix_a[i][j]=1.0+i;
          else
            {
             value+=1.0;
             matrix_a[i][j]=value/1e+20;
            }
         }
      }
//--- get another matrix
   matrix matrix_c = matrix_a * -1;

   ulong errors_epsilon=matrix_a.Compare(matrix_c,1e-15);
   ulong errors_digits=matrix_a.CompareByDigits(matrix_c,15);

   printf("Compare matrix %d x %d  errors_epsilon=%I64u  errors_digits=%I64u",size_m,


  /*
  Compare matrix 128 x 256  errors_epsilon=128  errors_digits=32768
  */
```

CompareEqual

Perform an absolute comparison of two matrices by unfolding successive rows into one-dimensional vectors.

```
ulong vector::Compare(
  const vector& vec     // vector to compare
   );

ulong matrix::CompareEqual(
  const matrix& mat      // matrix to compare
   );
```

Parameters

vec

[in] Vector to compare.

mat

[in] Matrix to compare.

Method description

Let us have two matrices: matrix A the method is called for and matrix B, which is passed as a method parameter. The comparison is performed as follows:

1. Matrices are expanded into one-dimensional vectors by successive concatenation of rows.
2. Vectors are compared element by element until the first mismatched element.
3. Depending on the comparison results, one of the values described below is returned.

Return Value

-1 — if the matrix A element is less than the corresponding matrix B element.

0 — if all elements of A and B matrices are identical.

1 — if the matrix A element exceeds the corresponding matrix B element.

Note

NaN value elements are taken into account when comparing.

NaN value elements are considered equal if they are present in both matrices at corresponding positions.

The NaN sign is not taken into account in the comparison.

An element with a NaN value is considered to be less than any other numeric value.

Example

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   matrix matrix_a= {{10, 3, 2}, {1, 8, 12}, {6, 5, 4}};
   matrix matrix_i=matrix::Identity(3, 3);
   matrix matrix_c=matrix_a.Inv();
   matrix matrix_check=matrix_a.MatMul(matrix_c);
   Print("matrix_check\n", matrix_check);

   ulong errors=matrix_check.Compare(matrix::Identity(3, 3), 1e-15);
   Print("errors=", errors);
   /*
   matrix_check
   [[1,0,0]
   [4.440892098500626e-16,1,8.881784197001252e-16]
   [4.440892098500626e-16,2.220446049250313e-16,0.9999999999999996]]
   errors=0
   */

//---  absolute comparison of matrices
   matrix<double> A = matrix_a;  // Matrix A initialization
   matrix<double> B = matrix_c;  // Matrix B initialization
   int result = A.CompareEqual(B);
   switch(result)
     {
      case -1:
         Print("Matrix A is smaller than matrix B");
         break;
      case 0:
         Print("Matrices A and B are identical");
         break;
      case 1:
         Print("Matrix A is greater than matrix B");
         break;
      case -2:
         Print("Error! Matrix A is not initialized");
         break;
      case 2:
         Print("Error! Matrix B is not initialized");
         break;
      case -3:
         Print("Error! The size of matrix A is less than the size of matrix B");
         break;
      case 3:
         Print("Error! The size of matrix A is greater than the size of matrix B");
         break;
      default:
         Print("Error! Unknown error");
```

```
            break;
        }
    }
```

Flat

Allows addressing a matrix element through one index instead of two.

```
bool matrix::Flat(
  const ulong    index,      // index
  const double   value       // value to set
   );

double matrix::Flat(
  const ulong    index,      // index
   );
```

Parameters

index

[in] Flat index

value

[in] Value to set by given index.

Return Value

Value by given index.

Note

For the matrix mat(3,3), access can be written as follows:

- reading: 'x=mat.Flat(4)', which is equivalent to 'x=mat[1][1]'
- writing: 'mat.Flat(5, 42)', equivalent to 'mat[1][2]=42'

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);
   ulong arg_max=matrix_a.ArgMax();
   Print("max_value=",matrix_a.Flat(arg_max));
   matrix_a.Flat(arg_max,0);
   arg_max=matrix_a.ArgMax();
   Print("max_value=",matrix_a.Flat(arg_max));


  /*
  matrix_a
  [[10,3,2]
   [1,8,12]
   [6,5,4]
   [7,11,9]]
  max_value=12.0
  max_value=11.0
  */
```

Clip

Limit the elements of a matrix/vector to a specified range of valid values.

```
bool matrix::Clip(
  const double  min_value,      // minimum value
  const double  max_value       // maximum value
   );
bool vector::Clip(
  const double  min_value,      // minimum value
  const double  max_value       // maximum value
   );
```

Parameters

min_value

[in] Minimum value.

max_value

[in] Maximum value.

Return Value

Returns true on success, false otherwise.

Note

The matrix (or vector) is processed in place. No copies are created.

Example

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   bool res=matrix_a.Clip(4,8);
   Print("matrix_a\n",matrix_a);

  /*
  matrix_a
  [[4,4,4]
   [4,5,6]
   [7,8,8]
   [8,8,8]]
  */
```

Reshape

Change the shape of a matrix without changing its data.

```
void  Reshape(
  const ulong  rows,      // new number or rows
  const ulong  cols       // new number or columns
   );
```

Parameters

rows

[in] New number or rows.

cols

[in] New number or columns.

Note

The matrix is processed in place. No copies are created. Any size can be specified, i.e., rows_new*cols_new!=rows_old*cols_old. When the matrix buffer is increased, the extra values are undefined.

Example

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

   Print("matrix_a\n",matrix_a);
   matrix_a.Reshape(2,6);
   Print("Reshape(2,6)\n",matrix_a);
   matrix_a.Reshape(3,5);
   Print("Reshape(3,5)\n",matrix_a);
   matrix_a.Reshape(2,4);
   Print("Reshape(2,4)\n",matrix_a);

  /*
  matrix_a
  [[1,2,3]
   [4,5,6]
   [7,8,9]
   [10,11,12]]
  Reshape(2,6)
  [[1,2,3,4,5,6]
   [7,8,9,10,11,12]]
  Reshape(3,5)
  [[1,2,3,4,5]
   [6,7,8,9,10]
   [11,12,0,3,0]]
  Reshape(2,4)
  [[1,2,3,4]
```

```
   [5,6,7,8]]
  */
```

Resize

Return a new matrix with a changed shape and size.

```
bool matrix::Resize(
  const ulong  rows,      // new number or rows
  const ulong  cols,      // new number or columns
  const ulong  reserve=0 // reserve amount in items
   );

bool vector::Resize(
  const ulong  size,      // new size.
  const ulong  reserve=0 // reserve amount in items.
   );
```

Parameters

rows

[in] New number or rows.

cols

[in] New number or columns.

Return Value

Returns true on success, false otherwise.

Note

The matrix (or vector) is processed in place. No copies are created. Any size can be specified, i.e., rows_new*cols_new!=rows_old*cols_old. Unlike Reshape, the matrix is processed row by row. When increasing the number of columns, the values of the extra columns are undefined. When increasing the number of rows, the values of elements in the new rows are undefined. When the number of columns is reduced, each row of the matrix is truncated.

Example

```
   matrix matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);
   matrix_a.Resize(2,6);
   Print("Ressize(2,6)\n",matrix_a);
   matrix_a.Resize(3,5);
   Print("Resize(3,5)\n",matrix_a);
   matrix_a.Resize(2,4);
   Print("Resize(2,4)\n",matrix_a);

   /*
   matrix_a
   [[1,2,3]
```

```
  [4,5,6]
  [7,8,9]
  [10,11,12]]
 Ressize(2,6)
 [[1,2,3,4,5,6]
  [4,5,6,10,11,12]]
 Resize(3,5)
 [[1,2,3,4,5]
  [4,5,6,10,11]
  [11,12,3,8,8]]
 Resize(2,4)
 [[1,2,3,4]
  [4,5,6,10]]
 */
```

Set

Sets the value for a vector element by the specified index.

```
bool vector::Set(
  ulong    index,      // element index
  double   value       // value
   );
```

Parameters

index

[in] Index of the element the value needs to be set for.

value

[in] Value.

Return Value

Returns true if successful, otherwise - false.

Note

The Set method does the same thing as assigning a value using square brackets, namely: *vector[index]=value*. The method has been added to simplify transferring a code from languages where this type of notation is used. The example below shows both options for filling the vector with values by the specified index.

Example:

```
void OnStart()
  {
//---
   vector v1(10, VectorAssignValues);
   Print("v1 = ", v1);

   vector v2(10, VectorSetValues);
   Print("v2 = ", v2);
  }
 /* Result
  v1 = [1,2,4,8,16,32,64,128,256,512]
  v2 = [1,2,4,8,16,32,64,128,256,512]
  */
//+------------------------------------------------------------------+
//| Fill a vector with powers of a number through the assignment operation  |
//+------------------------------------------------------------------+
void VectorAssignValues(vector& v, double initial=1)
  {
   double value=initial;
```

```
    for(ulong k=0; k<v.Size(); k++)
      {
       v[k]=value;
       value*=2;
      }
  }
//+------------------------------------------------------------------+
//| Fill a vector with powers of a number using the Set method        |
//+------------------------------------------------------------------+
void VectorSetValues(vector& v, double initial=1)
  {
    double value=initial;
    for(ulong k=0; k<v.Size(); k++)
      {
       v.Set(k, value);
       value*=2;
      }
  }
```

SwapRows

Swap rows in a matrix.

```
bool matrix::SwapRows(
  const ulong  row1,      // index of first row
  const ulong  row2       // index of second row
   );
```

Parameters

row1

[in] Index of the first row.

row2

[in] Index of the second row.

Return Value

Returns true on success, false otherwise.

Example

```
   matrix matrix_a={{1,2,3,4},
                    {5,6,7,8},
                    {9,10,11,12},
                    {13,14,15,16}};
   matrix matrix_i=matrix::Identity(4,4);
   matrix matrix_a1=matrix_a;
   matrix_a1.SwapRows(0,3);
   Print("matrix_a1\n",matrix_a1);

   matrix matrix_p=matrix_i;
   matrix_p.SwapRows(0,3);
   matrix matrix_c1=matrix_p.MatMul(matrix_a);
   Print("matrix_c1\n",matrix_c1);

  /*
  matrix_a1
  [[13,14,15,16]
   [5,6,7,8]
   [9,10,11,12]
   [1,2,3,4]]
  matrix_c1
  [[13,14,15,16]
   [5,6,7,8]
   [9,10,11,12]
   [1,2,3,4]]
  */
```

SwapCols

Swap columns in a matrix.

```
bool matrix::SwapCols(
  const ulong  row1,      // index of first column
  const ulong  row2       // index of second column
   );
```

Parameters

col1

[in] Index of the first column.

col2

[in] Index of the second column.

Return Value

Returns true on success, false otherwise.

Example

```
   matrix matrix_a={{1,2,3,4},
                    {5,6,7,8},
                    {9,10,11,12},
                    {13,14,15,16}};
   matrix matrix_i=matrix::Identity(4,4);
   matrix matrix_a1=matrix_a;
   matrix_a1.SwapCols(0,3);
   Print("matrix_a1\n",matrix_a1);

   matrix matrix_p=matrix_i;
   matrix_p.SwapCols(0,3);
   matrix matrix_c1=matrix_a.MatMul(matrix_p);
   Print("matrix_c1\n",matrix_c1);

  /*
  matrix_a1
  [[4,2,3,1]
   [8,6,7,5]
   [12,10,11,9]
   [16,14,15,13]]
  matrix_c1
  [[4,2,3,1]
   [8,6,7,5]
   [12,10,11,9]
   [16,14,15,13]]
  */
```

Split

Split a matrix into multiple submatrices.

```
bool matrix::Split(
  const ulong  parts,       // number of submatrices
  const int    axis,        // axis
  matrix&      splitted[]   // array of resulting submatrices
   );

void matrix::Split(
  const ulong& parts[],     // sizes of submatrices
  const int    axis,        // axis
  matrix&      splitted[]   // array of resulting submatrices
   );
```

Parameters

parts

[in] The number of submatrices to divide the matrix into.

axis

[in] Axis. 0 - horizontal axis, 1 - vertical axis.

splitted

[out] Array of resulting submatrices.

Return Value

Returns true on success, false otherwise.

Note

If the number of submatrices is specified, then same size submatrices are obtained. It means that the matrix size (0 - the number of rows, 1 - the number of columns) must be divisible by 'parts' without a remainder. Submatrices of different sizes can be obtained using an array of submatrix sizes. The elements of the size array are used until the entire matrix is divided. If the array of sizes has ended, and the matrix has not yet been completely divided, the undivided remainder will be the last submatrix.

Example

```
   matrix matrix_a={{ 1,  2,  3,  4,  5,  6},
                    { 7,  8,  9,10,11,12},
                    {13,14,15,16,17,18},
                    {19,20,21,22,23,24},
                    {25,26,27,28,29,30}};
   matrix splitted[];
   ulong  parts[]={2,2};
```

```
    bool res=matrix_a.Split(2,0,splitted);
    Print(res,"  ",GetLastError());
    ResetLastError();
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);

    res=matrix_a.Split(2,1,splitted);
    Print(res,"  ",GetLastError());
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);

    res=matrix_a.Split(parts,0,splitted);
    Print(res,"  ",GetLastError());
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);

   /*
   false  4003
   true  0
   splitted 0
   [[1,2,3]
    [7,8,9]
    [13,14,15]
    [19,20,21]
    [25,26,27]]
   splitted 1
   [[4,5,6]
    [10,11,12]
    [16,17,18]
    [22,23,24]
    [28,29,30]]
   true  0
   splitted 0
   [[1,2,3,4,5,6]
    [7,8,9,10,11,12]]
   splitted 1
   [[13,14,15,16,17,18]
    [19,20,21,22,23,24]]
   splitted 2
   [[25,26,27,28,29,30]]
   */
```

Hsplit

Split a matrix horizontally into multiple submatrices. Same as Split with axis=0

```
bool matrix::Hsplit(
  const ulong  parts,      // number of submatrices
  matrix&      splitted[]  // array of resulting submatrices
   );

void matrix::Hsplit(
  const ulong& parts[],    // sizes of submatrices
  matrix&      splitted[]  // array of resulting submatrices
   );
```

Parameters

parts

[in] The number of submatrices to divide the matrix into.

splitted

[out] Array of resulting submatrices.

Return Value

Returns true on success, false otherwise.

Note

If the number of submatrices is specified, then same size submatrices are obtained. It means that the number of rows must be divisible by 'parts' without a remainder. Submatrices of different sizes can be obtained using an array of submatrix sizes. The elements of the size array are used until the entire matrix is divided. If the array of sizes has ended, and the matrix has not yet been completely divided, the undivided remainder will be the last submatrix.

Example

```
   matrix matrix_a={{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9,10,11,12},
                    {13,14,15,16,17,18},
                    {19,20,21,22,23,24},
                    {25,26,27,28,29,30}};
   matrix splitted[];
   ulong  parts[]={2,4};

   bool res=matrix_a.Hsplit(2,splitted);
   Print(res,"  ",GetLastError());
   ResetLastError();
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);
```

```
    res=matrix_a.Hsplit(5,splitted);
    Print(res,"  ",GetLastError());
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);

    res=matrix_a.Hsplit(parts,splitted);
    Print(res,"  ",GetLastError());
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);


   /*
   false  4003
   true  0
   splitted 0
   [[1,2,3,4,5,6]]
   splitted 1
   [[7,8,9,10,11,12]]
   splitted 2
   [[13,14,15,16,17,18]]
   splitted 3
   [[19,20,21,22,23,24]]
   splitted 4
   [[25,26,27,28,29,30]]
   true  0
   splitted 0
   [[1,2,3,4,5,6]
   [7,8,9,10,11,12]]
   splitted 1
   [[13,14,15,16,17,18]
   [19,20,21,22,23,24]
   [25,26,27,28,29,30]]
   */
```

Vsplit

Split a matrix vertically into multiple submatrices. Same as Split with axis=1

```
bool matrix::Vsplit(
  const ulong  parts,      // number of submatrices
  matrix&      splitted[]  // array of resulting submatrices
   );

void matrix::Vsplit(
  const ulong& parts[],    // sizes of submatrices
  matrix&      splitted[]  // array of resulting submatrices
   );
```

Parameters

parts

[in] The number of submatrices to divide the matrix into.

splitted

[out] Array of resulting submatrices.

Return Value

Returns true on success, false otherwise.

Note

If the number of submatrices is specified, then same size submatrices are obtained. It means that the number of columns must be divisible by 'parts' without a remainder. Submatrices of different sizes can be obtained using an array of submatrix sizes. The elements of the size array are used until the entire matrix is divided. If the array of sizes has ended, and the matrix has not yet been completely divided, the undivided remainder will be the last submatrix.

Example

```
   matrix matrix_a={{ 1, 2, 3, 4, 5, 6},
                    { 7, 8, 9,10,11,12},
                    {13,14,15,16,17,18}};
   matrix splitted[];
   ulong  parts[]={2,3};

   matrix_a.Vsplit(2,splitted);
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);

   matrix_a.Vsplit(3,splitted);
   for(uint i=0; i<splitted.Size(); i++)
      Print("splitted ",i,"\n",splitted[i]);
```

```
    matrix_a.Vsplit(parts,splitted);
    for(uint i=0; i<splitted.Size(); i++)
       Print("splitted ",i,"\n",splitted[i]);


  /*
     splitted 0
     [[1,2,3]
      [7,8,9]
      [13,14,15]]
     splitted 1
     [[4,5,6]
      [10,11,12]
      [16,17,18]]

     splitted 0
     [[1,2]
      [7,8]
      [13,14]]
     splitted 1
     [[3,4]
      [9,10]
      [15,16]]
     splitted 2
     [[5,6]
      [11,12]
      [17,18]]

     splitted 0
     [[1,2]
      [7,8]
      [13,14]]
     splitted 1
     [[3,4,5]
      [9,10,11]
      [15,16,17]]
     splitted 2
     [[6]
      [12]
      [18]]

  */
```

ArgSort

Indirect sort of a matrix or vector.

```
vector vector::Sort(
   func_name   compare_func=NULL,   // comparison function
   T           context              // parameter for the custom sort function
   );

matrix matrix::Sort(
   func_name   compare_func=NULL    // comparison function
   T           context              // parameter for the custom sort function
   );

matrix matrix::Sort(
   const int   axis,                // axis for sorting
   func_name   compare_func=NULL    // comparison function
   T           context              // parameter for the custom sort function
   );
```

Parameters

axis

[in] The axing along which to sort: 0 is horizontal, 1 is vertical.

func_name

[in] Comparator. You can specify one of the values of the ENUM_SORT_MODE enumeration or your own comparison function. If no function is specified, ascending sort is used.

A custom comparison function can be of two types:

- int comparator(T x1,T x2)
- int comparator(T x1,T x2,TContext context)

Here T is the type of matrix or vector, and TContex is the type of the 'context' variable which is passed as an additional parameter to the Sort method.

context

[in] Additional optional parameter that can be passed to a custom sort function.

Return Value

Vector or matrix with the indexes of sorted elements. For example, the result [4,2,0,1,3] indicates that there should be an element with index 4 in the zero position, an element with index 2 in the first position, and so on.

Sort

Sort a matrix or vector in place.

```
void vector::Sort(
   func_name   compare_func=NULL,   // comparison function
   T           context              // parameter for the custom sort function
    );

void matrix::Sort(
   func_name   compare_func=NULL    // comparison function
   T           context              // parameter for the custom sort function
    );

void matrix::Sort(
   const int   axis,                // axis for sorting
   func_name   compare_func=NULL    // comparison function
   T           context              // parameter for the custom sort function
    );
```

Parameters

axis

[in] The axing along which to sort: 0 is horizontal, 1 is vertical.

func_name

[in] Comparator. You can specify one of the values of the ENUM_SORT_MODE enumeration or your own comparison function. If no function is specified, ascending sort is used.

A custom comparison function can be of two types:

- int comparator(T x1,T x2)
- int comparator(T x1,T x2,TContext context)

Here T is the type of matrix or vector, and TContex is the type of the 'context' variable which is passed as an additional parameter to the Sort method.

context

[in] Additional optional parameter that can be passed to a custom sort function.

Return Value

None. Sorting is performed in place, i.e. it is applied to the data of the matrix/vector for which the Sort method is called.

Example

```
//+------------------------------------------------------------------+
//| Sort function                                                    |
//+------------------------------------------------------------------+
int MyDoubleComparator(double x1,double x2,int sort_mode=0)
  {
    int res=x1<x2 ? -1 : (x1>x2 ? 1 : 0);
    return(sort_mode==0 ? res : -res);
   }
//+------------------------------------------------------------------+
//| Script start function                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- fill the vector
   vector v(100);
   //--- sort ascending
   v.Sort(MyDoubleComparator);   // an additional parameter with the default value '0
   Print(v);
   // sort descending
   v.Sort(MyDoubleComparator,1); // here the additional parameter '1' is explicitly s
   Print(v);
  }
```

Mathematical operations with matrices and vectors

Mathematical operations, including addition, subtraction, multiplication and division, can be performed on matrices and vectors element-wise.

Mathematical functions were originally designed to perform relevant operations on scalar values. Most of the functions can be applied to matrices and vectors. These include MathAbs, MathArccos, MathArcsin, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathLog10, MathMod, MathPow, MathRound, MathSin, MathSqrt, MathTan, MathExpm1, MathLog1p, MathArccosh, MathArcsinh, MathArctanh, MathCosh, MathSinh, and MathTanh. Such operations imply element-wise processing of matrices and vectors. Example

```
//---
  matrix a= {{1, 4}, {9, 16}};
  Print("matrix a=\n",a);
  a=MathSqrt(a);
  Print("MatrSqrt(a)=\n",a);
  /*
   matrix a=
   [[1,4]
    [9,16]]
   MatrSqrt(a)=
   [[1,2]
    [3,4]]
  */
```

For MathMod and MathPow, the second element can be either a scalar or a matrix/vector of the appropriate size.

The following example shows how to calculate the standard deviation by applying math functions to a vector.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- use the initializing function to fill the vector
  vector r(10, ArrayRandom); // array of random numbers from 0 to 1
//--- calculate the mean value
  double avr=r.Mean();       // array mean value
  vector d=r-avr;            // calculate an array of deviations from the mean
  Print("avr(r)=", avr);
  Print("r=", r);
  Print("d=", d);
  vector s2=MathPow(d, 2);   // array of squared deviations
  double sum=s2.Sum();       // sum of squared deviations
//--- calculate the standard deviation in two different methods
  double std=MathSqrt(sum/r.Size());
  Print(" std(r)=", std);
  Print("r.Std()=", r.Std());
 }
/*
  avr(r)=0.5300302133243813
  r=[0.8346201971495713,0.8031556138798182,0.6696676534318063,0.05386516922513505,0.54
  d=[0.30458998382519,0.2731254005554369,0.1396374401074251,-0.4761650440992462,0.019
   std(r)=0.2838269732183663
  r.Std()=0.2838269732183663
*/
//+------------------------------------------------------------------+
//| Fills a vector with random values                                |
//+------------------------------------------------------------------+
void ArrayRandom(vector& v)
 {
  for(ulong i=0; i<v.Size(); i++)
    v[i]=double(MathRand())/32767.;
 }
```

Mathematical operations

Mathematical operations, including addition, subtraction, multiplication and division, can be performed on matrices and vectors element-wise.

Both matrices or both vectors must be of the same type and must have the same dimensions. Each element of the matrix operates on the corresponding element of the second matrix.

You can also use a scalar of the appropriate type (double, float or complex) as the second term (multiplier, subtrahend or divisor). In this case, each member of the matrix or vector will operate on the specified scalar.

```
   matrix matrix_a={{0.1,0.2,0.3},{0.4,0.5,0.6}};
   matrix matrix_b={{1,2,3},{4,5,6}};

   matrix matrix_c1=matrix_a+matrix_b;
   matrix matrix_c2=matrix_b-matrix_a;
   matrix matrix_c3=matrix_a*matrix_b;   // Hadamard product, not to be confused with 
   matrix matrix_c4=matrix_b/matrix_a;

   matrix_c1=matrix_a+1;
   matrix_c2=matrix_b-double_value;
   matrix_c3=matrix_a*M_PI;
   matrix_c4=matrix_b/0.1;

//--- operations in place are possible
   matrix_a+=matrix_b;
   matrix_a/=2;
```

The same operations are available for vectors.

Mathematical functions

The following mathematical functions can be applied to matrices and vectors: MathAbs, MathArccos, MathArcsin, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathLog10, MathMod, MathPow, MathRound, MathSin, MathSqrt, MathTan, MathExpm1, MathLog1p, MathArccosh, MathArcsinh, MathArctanh, MathCosh, MathSinh, MathTanh. Such operations imply element-wise processing of matrices and vectors.

For MathMod and MathPow, the second element can be either a scalar or a matrix/vector of the appropriate size.

```
   matrix<T> mat1(128,128);
   matrix<T> mat3(mat1.Rows(),mat1.Cols());
   ulong     n,size=mat1.Rows()*mat1.Cols();
...
   mat2=MathPow(mat1,(T)1.9);
   for(n=0; n<size; n++)
     {
      T res=MathPow(mat1.Flat(n),(T)1.9);
      if(res!=mat2.Flat(n))
         errors++;
     }
   mat2=MathPow(mat1,mat3);
   for(n=0; n<size; n++)
     {
      T res=MathPow(mat1.Flat(n),mat3.Flat(n));
      if(res!=mat2.Flat(n))
         errors++;
     }
...
   vector<T> vec1(16384);
   vector<T> vec3(vec1.Size());
   ulong     n,size=vec1.Size();
...
   vec2=MathPow(vec1,(T)1.9);
   for(n=0; n<size; n++)
     {
      T res=MathPow(vec1[n],(T)1.9);
      if(res!=vec2[n])
         errors++;
     }
   vec2=MathPow(vec1,vec3);
   for(n=0; n<size; n++)
     {
      T res=MathPow(vec1[n],vec3[n]);
      if(res!=vec2[n])
         errors++;
     }
```

Matrix and vector products

Matrix and vector product calculations include:

- matrix multiplication
- vector multiplication
- computing covariance matrix
- computing the cross-correlation of two vectors
- computing the convolution of two vectors
- computing the correlation coefficient

Function	Action
MatMul	Matrix product of two matrices
GeMM	General matrix multiplication (GeMM)
Power	Raise a square matrix to the integer power
Dot	Dot product of two vectors
Kron	Return Kronecker product of two matrices, matrix and vector, vector and matrix or two vectors
Inner	Inner product of two matrices
Outer	Compute the outer product of two matrices or two vectors
CorrCoef	Compute the Pearson correlation coefficient (linear correlation coefficient)
Cov	Compute the covariance matrix
Correlate	Compute the cross-correlation of two vectors
Convolve	Return the discrete, linear convolution of two vectors

MatMul

The MatMul method, which enables the multiplication of matrices and vectors, has several overloads.

Multiplying a matrix by a matrix: `matrix[M][K] * matrix[K][N] = matrix[M][N]`

```
matrix  matrix::MatMul(
   const matrix&   b       // second matrix
    );
```

Multiplying a vector by a matrix: `horizontal vector[K] * matrix[K][N] = horizontal vector[N]`

```
vector  vector::MatMul(
   const matrix&   b       // matrix
    );
```

Multiplying a matrix by a vector: `matrix[M][K] * vertical vector[K] = vertical vector[M]`

```
vector  matrix::MatMul(
   const vector&   b       // vector
    );
```

Scalar vector multiplication: `horizontal vector * vertical vector = dot value`

```
scalar  vector::MatMul(
   const vector&   b       // second vector
    );
```

Parameters

b

[in] Matrix or vector.

Return Value

Matrix, vector, or scalar, depending on the method used.

Note

The matrices should be compatible for multiplication, i.e. the number of columns in the first matrix should be equal to the number of rows in the second matrix. Matrix multiplication is non-commutative: the result of multiplying the first matrix by the second one is not equal to the result of multiplying the second matrix by the first one in the general case.

The matrix product consists of all possible combinations of scalar products of the row vectors of the first matrix and the column vectors of the second matrix.

In scalar multiplication, vectors must have the same length.

When multiplying a vector and a matrix, the length of the vector must exactly match the number of columns in the matrix.

Naive matrix multiplication algorithm in MQL5:

```
matrix MatrixProduct(const matrix& matrix_a, const matrix& matrix_b)
  {
   matrix matrix_c;

   if(matrix_a.Cols()!=matrix_b.Rows())
      return(matrix_c);

   ulong M=matrix_a.Rows();
   ulong K=matrix_a.Cols();
   ulong N=matrix_b.Cols();
   matrix_c=matrix::Zeros(M,N);

   for(ulong m=0; m<M; m++)
      for(ulong k=0; k<K; k++)
         for(ulong n=0; n<N; n++)
            matrix_c[m][n]+=matrix_a[m][k]*matrix_b[k][n];

   return(matrix_c);
  }
```

Matrix multiplication example

```
   matrix a={{1, 0, 0},
             {0, 1, 0}};
   matrix b={{4, 1},
             {2, 2},
             {1, 3}};
   matrix c1=a.MatMul(b);
   matrix c2=b.MatMul(a);
   Print("c1 = \n", c1);
   Print("c2 = \n", c2);
/*
   c1 = 
   [[4,1]
    [2,2]]
   c2 = 
   [[4,1,0]
    [2,2,0]
    [1,3,0]]
*/
```

An example of multiplying a horizontal vector by a matrix

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create a 3x5 matrix
   matrix m35;
   m35.Init(3, 5, Arange);
//---
   vector v3 = {1, 2, 3};
   Print("Product of horizontal vector v and matrix m[3,5]");
   Print("On the left, vector v3 = ", v3);
   Print("On the right, matrix m35 = \n", m35);
   Print("v3.MatMul(m35) = horizontal vector v[5] \n", v3.MatMul(m35));

  /* Result
    Product of horizontal vector v3 and matrix m[3,5]
    On the left, vector v3 = [1,2,3]
    On the right, matrix m35 =
    [[0,1,2,3,4]
     [5,6,7,8,9]
     [10,11,12,13,14]]
    v3.MatMul(m35) = horizontal vector v[5]
    [40,46,52,58,64]
   */
  }
//+------------------------------------------------------------------+
//|  Fill the matrix with increasing values                          |
//+------------------------------------------------------------------+
void Arange(matrix & m, double start = 0, double step = 1)
  {
//---
   ulong cols = m.Cols();
   ulong rows = m.Rows();
   double value = start;
   for(ulong r = 0; r < rows; r++)
     {
      for(ulong c = 0; c < cols; c++)
        {
         m[r][c] = value;
         value += step;
        }
     }
//---
  }
```

An example of how to multiply a matrix by a vertical vector

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create a 3x5 matrix
   matrix m35;
   m35.Init(3, 5, Arange);
//---
   Print("Product of matrix m[3,5] and vertical vector v[5]");
   vector v5 = {1,2,3,4,5};
   Print("On the left, m35 = \n",m35);
   Print("On the right v5 = ",v5);
   Print("m35.MatMul(v5) = vertical vector v[3] \n",m35.MatMul(v5));

  /* Result
   Product of matrix m[3,5] and vertical vector v[5]
   On the left, m35 =
   [[0,1,2,3,4]
    [5,6,7,8,9]
    [10,11,12,13,14]]
   On the right, v5 = [1,2,3,4,5]
   m35.MatMul(v5) = vertical vector v[3]
   [40,115,190]
   */
  }
//+------------------------------------------------------------------+
//|  Fill the matrix with increasing values                          |
//+------------------------------------------------------------------+
void Arange(matrix & m, double start = 0, double step = 1)
  {
//---
   ulong cols = m.Cols();
   ulong rows = m.Rows();
   double value = start;
   for(ulong r = 0; r < rows; r++)
     {
      for(ulong c = 0; c < cols; c++)
        {
         m[r][c] = value;
         value += step;
        }
     }
//---
  }
```

An example of scalar (dot) product of vectors

```
void OnStart()
  {
```

```
//--- scalar product of a horizontal vector and a vertical one
   vector a= {1, 2, 3};  // horizontal vector
   vector b= {4, 5, 6};  // vertical vector
   Print("a = ", a);
   Print("b = ", b);
   Print("1) a.MatMul(b) = ", a.MatMul(b));
   //--- see that the Dot method generates the same result
   Print("2) a.Dot(b) = ", a.Dot(b));

  /* Result
   a = [1,2,3]
   b = [4,5,6]
   1) a.MatMul(b) = 32.0
   2) a.Dot(b) = 32.0
    */
  }
```

See also

Dot, GeMM

GeMM

The GeMM (General Matrix Multiply) method implements the general multiplication of two matrices. The operation is defined as $C \leftarrow \alpha A B + \beta C$, where A and B matrices can be optionally transposed. With a normal multiplication of matrices AB (MatMul), the *alpha* scalar is assumed to be equal to 1 and *beta* equal to zero.

The main difference between GeMM and MatMul in terms of efficiency is that MatMul always creates a new matrix/vector object, while GeMM works with an existing matrix object and does not re-create it. Therefore, when you use GeMM and pre-allocate memory for the corresponding matrix, then, while working with the same matrix sizes, there will be no memory reallocations. This can be an important advantage of GeMM for bulk computing, for example, when running optimizations in a strategy tester or when training a neural network.

Similar to MatMul, GeMM also has 4 overloads. However, the semantics of the fourth overload has been modified in order to enable the multiplication of a vertical vector with a and horizontal one.

In an existing matrix/vector object, it is not necessary to pre-allocate memory. The memory will be allocated and filled with zeros at the first GeMM call.

Multiplying a matrix by a matrix: `matrix C[M][N]` = α * (`matrix A[M][K]  *  matrix B[K][N])` + β * `matrix C[M][N]`

```
bool  matrix::GeMM(
   const matrix &A,    // first matrix
   const matrix &B,    // second matrix
   double alpha,       // alpha multiplier for product AB
   double beta,        // beta multiplier for matrix C
   uint   flags        // a combination of ENUM_GEMM values (bitwise OR), which determ
    );
```

Multiplying a vector by a matrix: `vector C[N]` = α * (`vector A[K]  *  matrix B[K][N])` + β * `vector C[N]`

```
bool  vector::GeMM(
   const vector &A,    // horizontal vector
   const matrix &B,    // matrix
   double alpha,       // alpha multiplier for product AB
   double beta,        // beta multiplier for vector C
   uint   flags        // ENUM_GEMM enumeration value that determines whether matrix A
    );
```

Multiplying a matrix by a vector: `vector C[M]` = α * (`matrix A[M][K]` * `vector B[K]  *  )` + β * `vector C[M]`

```
bool  vector::GeMM(
   const matrix &A,    // matrix
   const vector &B,    // vertical vector
   double alpha,       // alpha multiplier for product AB
   double beta,        // beta multiplier for vector C
   uint   flags        // ENUM_GEMM enumeration value that determines whether matrix B
    );
```

Multiplying a vector by a vector: matrix C[M][N] = α * (vector A[M] * vector B[N] *) + β * matrix C[M][N]. This overload returns a matrix, unlike MatMul where it returns a scalar.

```
bool  matrix::GeMM(
   const vector &A,     // first vector
   const vector &B,     // second vector
   double alpha,        // alpha multiplier for product AB
   double beta,         // beta for matrix C
   uint   flags         // ENUM_GEMM enumeration value that determines whether matrix C
    );
```

Parameters

A

[in] Matrix or vector.

B

[in] Matrix or vector.

alpha

[in] Alpha multiplier for the AB product.

beta

[in] Beta multiplier for the resulting C matrix.

flags

[in] The ENUM_GEMM enumeration value that determines whether matrices A, B and C are transposed.

Return Value

Returns true on success or false otherwise.

ENUM_GEMM

Enumeration of flags for the GeMM method.

ID	Description
TRANSP_A	Use transposed matrix A
TRANSP_B	Use transposed matrix B
TRANSP_C	Use transposed matrix C

Note

Matrices and vectors of float, double and complex types can be used as parameters A and B. The template variants of the GeMM method are as follows:

```
bool matrix<T>::GeMM(const matrix<T> &A,const matrix<T> &B,T alpha,T beta,ulong flags)
bool matrix<T>::GeMM(const vector<T> &A,const vector<T> &B,T alpha,T beta,ulong flags)

bool vector<T>::GeMM(const vector<T> &A,const matrix<T> &B,T alpha,T beta,ulong flags)
bool vector<T>::GeMM(const matrix<T> &A,const vector<T> &B,T alpha,T beta,ulong flags)
```

Basically the general matrix multiplication function is described as:

```
C[m,n] = α *Sum(A[m,k]*B[k,n]) + β*C[m,n]
```

with the following sizes: matrix A is M x K, matrix B is K x N and matrix C is M x N.

Thus, the matrices should be compatible for multiplication, i.e. the number of columns in the first matrix should be equal to the number of rows in the second matrix. Matrix multiplication is non-commutative: the result of multiplying the first matrix by the second one is not equal to the result of multiplying the second matrix by the first one in the general case.

Example:

```
void OnStart()
  {
   vector vector_a= {1, 2, 3, 4, 5};
   vector vector_b= {4, 3, 2, 1};
   matrix matrix_c;
//--- calculate GeMM for two vectors
   matrix_c.GeMM(vector_a, vector_b, 1, 0);
   Print("matrix_c:\n ", matrix_c, "\n");
   /*
   matrix_c:
    [[4,3,2,1]
    [8,6,4,2]
    [12,9,6,3]
    [16,12,8,4]
    [20,15,10,5]]
   */
//--- create matrices as vectors
   matrix matrix_a(5, 1);
   matrix matrix_b(1, 4);
   matrix_a.Col(vector_a, 0);
   matrix_b.Row(vector_b, 0);
   Print("matrix_a:\n ", matrix_a);
   Print("matrix_b:\n ", matrix_b);
   /*
   matrix_a:
   [[1]
   [2]
   [3]
   [4]
```

```
    [5]]
   matrix_b:
    [[4,3,2,1]]
    */
//-- calculate GeMM for two matrices and get the same result
   matrix_c.GeMM(matrix_a, matrix_b, 1, 0);
   Print("matrix_c:\n ", matrix_c);
   /*
   matrix_c:
    [[4,3,2,1]
    [8,6,4,2]
    [12,9,6,3]
    [16,12,8,4]
    [20,15,10,5]]
   */
  }
```

See also

[MatMul]

Power

Raise a square matrix to the integer power.

```
matrix matrix::Power(
  const int  power      // power
   );
```

Parameters

power

[in] The exponent can be any integer, positive, negative, or zero.

Return Value

Matrix.

Note

The resulting matrix has the same size as the original matrix. In raised to the power of 0, the identity matrix is returned. The positive power n means that the original matrix is multiplied n times by itself. The negative power -n means that the original matrix is first inverted, and then the inverted matrix is multiplied by itself n times.

A simple algorithm for raising a matrix to a power in MQL5:

```
bool MatrixPower(matrix& c, const matrix& a, const int power)
  {
//--- matrix must be square
   if(a.Rows()!=a.Cols())
      return(false);
//--- the size of the resulting matrix is exactly the same
   ulong  rows=a.Rows();
   ulong  cols=a.Cols();
   matrix result(rows,cols);
//--- when raised to zero, return the identity matrix
   if(power==0)
      result.Identity();
   else
     {
      //--- for a negative degree, first invert the matrix
      if(power<0)
        {
         matrix inverted=a.Inv();
         result=inverted;
         for(int i=-1; i>power; i--)
            result=result.MatMul(inverted);
        }
      else
        {
         result=a;
         for(int i=1; i<power; i++)
            result=result.MatMul(a);
        }
     }
//---
   c=result;
   return(true);
  }
```

MQL5 example:

```
  matrix i= {{0, 1}, {-1, 0}};
  Print("i:\n", i);

  Print("i.Power(3):\n", i.Power(3));

  Print("i.Power(0):\n", i.Power(0));

  Print("i.Power(-3):\n", i.Power(-3));

  /*
  i:
  [[0,1]
   [-1,0]]

  i.Power(3):
  [[0,-1]
   [1,0]]

  i.Power(0):
  [[1,0]
   [0,1]]

  i.Power(-3):
  [[0, -1]
   [1,0]]
  */
```

Python example:

```
import numpy as np
from numpy.linalg import matrix_power

# matrix equiv. of the imaginary unit
```

```
i = np.array([[0, 1], [-1, 0]])
print("i:\n",i)

# should = -i
print("matrix_power(i, 3) :\n",matrix_power(i, 3) )

print("matrix_power(i, 0):\n",matrix_power(i, 0))

# should = 1/(-i) = i, but w/ f.p. elements
print("matrix_power(i, -3):\n",matrix_power(i, -3))

i:
 [[ 0  1]
 [-1  0]]

matrix_power(i, 3) :
 [[ 0 -1]
 [ 1  0]]

matrix_power(i, 0):
 [[1 0]
 [0 1]]

matrix_power(i, -3):
 [[ 0.  1.]
 [-1.  0.]]
```

Dot

Dot product of two vectors.

```
double vector::Dot(
   const vector&  b       // second vector
    );
```

Parameters

b

[in] Vector.

Return Value

Scalar.

Note

The dot product of two matrices is the matrix product matrix::MatMul().

A simple algorithm for the scalar product of vectors in MQL5:

```
double VectorDot(const vector& vector_a, const vector& vector_b)
  {
   double dot=0;

   if(vector_a.Size()==vector_b.Size())
     {
      for(ulong i=0; i<vector_a.Size(); i++)
         dot+=vector_a[i]*vector_b[i];
     }

   return(dot);
  }
```

MQL5 example:

```
   for(ulong i=0; i<rows; i++)
     {
      vector v1=a.Row(i);
      for(ulong j=0; j<cols; j++)
        {
         vector v2=b.Row(j);
         result[i][j]=v1.Dot(v2);
        }
     }
```

Python example:

```
import numpy as np

a = [1, 0, 0, 1]
b = [4, 1, 2, 2]
print(np.dot(a, b))

>>> 6
```

Kron

Return Kronecker product of two matrices, matrix and vector, vector and matrix or two vectors.

```
matrix matrix::Kron(
  const matrix&  b      // second matrix
   );

matrix matrix::Kron(
  const vector&  b      // vector
   );

matrix vector::Kron(
  const matrix&  b      // matrix
   );

matrix vector::Kron(
  const vector&  b      // second vector
   );
```

Parameters

b

[in] second matrix.

Return Value

Matrix.

Note

The Kronecker product is also referred to as the block matrix multiplication.

A simple algorithm for the Kronecker product for two matrices in MQL5:

```
matrix MatrixKronecker(const matrix& matrix_a,const matrix& matrix_b)
  {
   ulong  M=matrix_a.Rows();
   ulong  N=matrix_a.Cols();
   ulong  P=matrix_b.Rows();
   ulong  Q=matrix_b.Cols();
   matrix matrix_c(M*P,N*Q);

   for(ulong m=0; m<M; m++)
      for(ulong n=0; n<N; n++)
         for(ulong p=0; p<P; p++)
            for(ulong q=0; q<Q; q++)
```

```
                matrix_c[m*P+p][n*Q+q]=matrix_a[m][n] * matrix_b[p][q];

   return(matrix_c);
  }
```

MQL5 example:

```
   matrix a={{1,2,3},{4,5,6}};
   matrix b=matrix::Identity(2,2);
   vector v={1,2};

   Print(a.Kron(b));
   Print(a.Kron(v));

  /*
   [[1,0,2,0,3,0]
    [0,1,0,2,0,3]
    [4,0,5,0,6,0]
    [0,4,0,5,0,6]]

   [[1,2,2,4,3,6]
    [4,8,5,10,6,12]]
  */
```

Python example:

```
import numpy as np

A = np.arange(1,7).reshape(2,3)
B = np.identity(2)
V = [1,2]
print(np.kron(A, B))
print("")
print(np.kron(A, V))

[[1. 0. 2. 0. 3. 0.]
 [0. 1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6. 0.]
 [0. 4. 0. 5. 0. 6.]]

[[ 1  2  2  4  3  6]
 [ 4  8  5 10  6 12]]
```

Inner

Inner product of two matrices.

```
matrix matrix::Inner(
  const matrix&  b       // second matrix
   );
```

Parameters

b

[in] Matrix.

Return Value

Matrix.

Note

The inner product for two vectors is the dot product of the two vectors vector::Dot().

A simple algorithm for the inner product of two matrices in MQL5:

```
bool MatrixInner(matrix& c, const matrix& a, const matrix& b)
  {
//--- the number of columns must equal
   if(a.Cols()!=b.Cols())
      return(false);
//--- size of the resulting matrix depends on the number of vectors in each of the mat
   ulong  rows=a.Rows();
   ulong  cols=b.Rows();
   matrix result(rows,cols);
//---
   for(ulong i=0; i<rows; i++)
     {
      vector v1=a.Row(i);
      for(ulong j=0; j<cols; j++)
        {
         vector v2=b.Row(j);
         result[i][j]=v1.Dot(v2);
        }
     }
//---
   c=result;
   return(true);
  }
```

MQL5 example:

```
   matrix a={{0,1,2},{3,4,5}};
   matrix b={{0,1,2},{3,4,5},{6,7,8}};
   matrix c=a.Inner(b);
   Print(c);
   matrix a1={{0,1,2}};
   matrix c1=a1.Inner(b);
   Print(c1);

  /*
  [[5,14,23]
  [14,50,86]]
  [[5,14,23]]
  */
```

Python example:

```
import numpy as np

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
A1= np.arange(3)
print(np.inner(A, B))
print("");
print(np.inner(A1, B))

import numpy as np

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
A1= np.arange(3)
print(np.inner(A, B))
print("");
print(np.inner(A1, B))
```

Outer

Compute the outer product of two matrices or two vectors.

```
matrix matrix::Outer(
  const matrix&  b       // second matrix
   );

matrix vector::Outer(
  const vector&  b       // second vector
   );
```

Parameters

b

[in] Matrix.

Return Value

Matrix.

Note

The outer product, like the Kronecker product, is also a block matrix (and vector) multiplication.

A simple algorithm for the outer product of two matrices in MQL5:

```
matrix MatrixOuter(const matrix& matrix_a, const matrix& matrix_b)
  {
//--- size of the resulting matrix depends on the matrix sizes
   ulong  rows=matrix_a.Rows()*matrix_a.Cols();
   ulong  cols=matrix_b.Rows()*matrix_b.Cols();
   matrix matrix_c(rows,cols);
   ulong  cols_a=matrix_a.Cols();
   ulong  cols_b=matrix_b.Cols();
//---
   for(ulong i=0; i<rows; i++)
     {
      ulong row_a=i/cols_a;
      ulong col_a=i%cols_a;
      for(ulong j=0; j<cols; j++)
        {
         ulong row_b=j/cols_b;
         ulong col_b=j%cols_b;
         matrix_c[i][j]=matrix_a[row_a][col_a] * matrix_b[row_b][col_b];
        }
     }
//---
   return(matrix_c);
  }
```

MQL5 example:

```
   vector vector_a={0,1,2,3,4,5};
   vector vector_b={0,1,2,3,4,5,6};
   Print("vector_a.Outer\n",vector_a.Outer(vector_b));
   Print("vector_a.Kron\n",vector_a.Kron(vector_b));

   matrix matrix_a={{0,1,2},{3,4,5}};
   matrix matrix_b={{0,1,2},{3,4,5},{6,7,8}};
   Print("matrix_a.Outer\n",matrix_a.Outer(matrix_b));
   Print("matrix_a.Kron\n",matrix_a.Kron(matrix_b));

  /*
   vector_a.Outer
   [[0,0,0,0,0,0,0]
    [0,1,2,3,4,5,6]
    [0,2,4,6,8,10,12]
    [0,3,6,9,12,15,18]
    [0,4,8,12,16,20,24]
    [0,5,10,15,20,25,30]]
   vector_a.Kron
   [[0,0,0,0,0,0,0,0,1,2,3,4,5,6,0,2,4,6,8,10,12,0,3,6,9,12,15,18,0,4,8,12,16,20,24,0,
   matrix_a.Outer
   [[0,0,0,0,0,0,0,0,0]
    [0,1,2,3,4,5,6,7,8]
    [0,2,4,6,8,10,12,14,16]
    [0,3,6,9,12,15,18,21,24]
    [0,4,8,12,16,20,24,28,32]
    [0,5,10,15,20,25,30,35,40]]
   matrix_a.Kron
   [[0,0,0,0,1,2,0,2,4]
    [0,0,0,3,4,5,6,8,10]
    [0,0,0,6,7,8,12,14,16]
    [0,3,6,0,4,8,0,5,10]
    [9,12,15,12,16,20,15,20,25]
    [18,21,24,24,28,32,30,35,40]]
   */
```

Python example:

```
import numpy as np

A = np.arange(6)
```

```
B = np.arange(7)
print("np.outer")
print(np.outer(A, B))
print("np.kron")
print(np.kron(A, B))

A = np.arange(6).reshape(2, 3)
B = np.arange(9).reshape(3, 3)
print("np.outer")
print(np.outer(A, B))
print("np.kron")

np.outer
[[ 0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6]
 [ 0  2  4  6  8 10 12]
 [ 0  3  6  9 12 15 18]
 [ 0  4  8 12 16 20 24]
 [ 0  5 10 15 20 25 30]]
np.kron
[ 0  0  0  0  0  0  0  0  1  2  3  4  5  6  0  2  4  6  8 10 12  0  3  6
  9 12 15 18  0  4  8 12 16 20 24  0  5 10 15 20 25 30]
np.outer
[[ 0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8]
 [ 0  2  4  6  8 10 12 14 16]
 [ 0  3  6  9 12 15 18 21 24]
 [ 0  4  8 12 16 20 24 28 32]
 [ 0  5 10 15 20 25 30 35 40]]
np.kron
[[ 0  0  0  0  1  2  0  2  4]
 [ 0  0  0  3  4  5  6  8 10]
 [ 0  0  0  6  7  8 12 14 16]
 [ 0  3  6  0  4  8  0  5 10]
 [ 9 12 15 12 16 20 15 20 25]
 [18 21 24 24 28 32 30 35 40]]
```

CorrCoef

Compute the Pearson correlation coefficient (linear correlation coefficient).

```
matrix matrix::CorrCoef(
  const bool     rowvar=true  // rows or cols vectors of observations
);

scalar vector::CorrCoef(
  const vector&  b            // second vector
);
```

Return Value

Pearson product-moment correlation coefficients.

Note

The correlation coefficient is in the range [-1, 1].

Due to floating point rounding the resulting array may not be Hermitian, the diagonal elements may not be 1, and the elements may not satisfy the inequality abs(a) <= 1. The real and imaginary parts are clipped to the interval [-1, 1] in an attempt to improve on that situation but is not much help in the complex case.

A simple algorithm for calculating the correlation coefficient of two vectors using MQL5:

```
double VectorCorrelation(const vector& vector_x,const vector& vector_y)
  {
   ulong n=vector_x.Size()<vector_y.Size() ? vector_x.Size() : vector_y.Size();
   if(n<=1)
      return(0);

   ulong  i;
   double xmean=0;
   double ymean=0;
   for(i=0; i<n; i++)
     {
      if(!MathIsValidNumber(vector_x[i]))
         return(0);
      if(!MathIsValidNumber(vector_y[i]))
         return(0);
      xmean+=vector_x[i];
      ymean+=vector_y[i];
     }
   xmean/=(double)n;
   ymean/=(double)n;
   double s=0;
   double xv=0;
   double yv=0;
   double t1=0;
   double t2=0;
//--- calculation
   s=0;
   for(i=0; i<n; i++)
     {
      t1=vector_x[i]-xmean;
      t2=vector_y[i]-ymean;
      xv+=t1*t1;
      yv+=t2*t2;
      s+=t1*t2;
     }
//--- check
   if(xv==0 || yv==0)
      return(0);
//--- return result
   return(s/(MathSqrt(xv)*MathSqrt(yv)));
  }
```

MQL5 example:

```
   vectorf vector_a={1,2,3,4,5};
   vectorf vector_b={0,1,0.5,2,2.5};
   Print("vectors correlation ",vector_a.CorrCoef(vector_b));
//---
   matrixf matrix_a={{1,2,3,4,5},
                     {0,1,0.5,2,2.5}};
   Print("matrix rows correlation\n",matrix_a.CorrCoef());
   matrixf matrix_a2=matrix_a.Transpose();
   Print("transposed matrix cols correlation\n",matrix_a2.CorrCoef(false));
   matrixf matrix_a3={{1.0f, 2.0f, 3.0f, 4.0f, 5.0f},
                      {0.0f, 1.0f, 0.5f, 2.0f, 2.5f},
                      {0.1f, 1.0f, 2.0f, 1.0f, 0.3f}};
   Print("rows correlation\n",matrix_a3.CorrCoef());
   Print("cols correlation\n",matrix_a3.CorrCoef(false));

  /*
   vectors correlation 0.9149913787841797
   matrix rows correlation
   [[1,0.91499138]
    [0.91499138,1]]
   transposed matrix cols correlation
   [[1,0.91499138]
    [0.91499138,1]]
    rows correlation
    [[1,0.91499138,0.08474271]
     [0.91499138,1,-0.17123166]
     [0.08474271,-0.17123166,1]]
    cols correlation
    [[1,0.99587059,0.85375023,0.91129309,0.83773589]
     [0.99587059,1,0.80295509,0.94491106,0.88385159]
     [0.85375023,0.80295509,1,0.56362146,0.43088508]
     [0.91129309,0.94491106,0.56362146,1,0.98827404]
     [0.83773589,0.88385159,0.43088508,0.98827404,1]]
    */
```

Python example:

```
import numpy as np
va=[1,2,3,4,5]
vb=[0,1,0.5,2,2.5]
print("vectors correlation")
print(np.corrcoef(va,vb))

ma=np.zeros((2,5))
ma[0,:]=va
ma[1,:]=vb
print("matrix rows correlation")
print(np.corrcoef(ma))
print("transposed matrix cols correlation")
print(np.corrcoef(np.transpose(ma),rowvar=False))
print("")

ma1=[[1,2,3,4,5],[0,1,0.5,2,2.5],[0.1,1,0.2,1,0.3]]
print("rows correlation\n",np.corrcoef(ma1))
print("cols correlation\n",np.corrcoef(ma1,rowvar=False))

transposed matrix cols correlation
[[1.         0.91499142]
 [0.91499142 1.        ]]

rows correlation
 [[1.         0.91499142 0.1424941 ]
 [0.91499142 1.         0.39657517]
 [0.1424941  0.39657517 1.        ]]
cols correlation
 [[1.         0.99587059 0.98226063 0.91129318 0.83773586]
 [0.99587059 1.         0.99522839 0.94491118 0.88385151]
 [0.98226063 0.99522839 1.         0.97234063 0.92527551]
 [0.91129318 0.94491118 0.97234063 1.         0.98827406]
 [0.83773586 0.88385151 0.92527551 0.98827406 1.        ]]
```

Cov

Compute the covariance matrix.

```
matrix matrix::Cov(
  const bool    rowvar=true  // rows or cols vectors of observations
);

matrix matrix::Cov(
  const bool    rowvar,      // rows or cols vectors of observations
  const int     ddof         // delta degrees of freedom
);

matrix vector::Cov(
  const vector& b            // second vector
);

matrix vector::Cov(
  const vector& b,           // second vector
  const int     ddof         // delta degrees of freedom
);
```

Parameters

rowwar

[in] If rowvar is true (default), then each row represents a variable, with observations in the columns. Otherwise, the relationship is transposed: each column represents a variable, while the rows contain observations.

b

[in] Second vector of observations.

ddof

[in] “Delta Degrees of Freedom”: the divisor used in the calculation is N - ddof, where N represents the number of elements. By default ddof is 1.

Note

Compute the covariance matrix.

A simple algorithm for calculating the covariance matrix of two vectors using MQL5:

```
bool VectorCovariation(const vector& vector_a,const vector& vector_b,matrix& matrix_c)
  {
   int i,j;
   int m=2;
   int n=(int)(vector_a.Size()<vector_b.Size()?vector_a.Size():vector_b.Size());
//--- checks
   if(n<=1)
      return(false);
   for(i=0; i<n; i++)
     {
      if(!MathIsValidNumber(vector_a[i]))
         return(false);
      if(!MathIsValidNumber(vector_b[i]))
         return(false);
     }
//---
   matrix matrix_x(2,n);
   matrix_x.Row(vector_a,0);
   matrix_x.Row(vector_b,1);
   vector t=vector::Zeros(m);
//--- calculation
   for(i=0; i<m; i++)
      for(j=0; j<n; j++)
         t[i]+=matrix_x[i][j]/double(n);
   for(i=0; i<m; i++)
      for(j=0; j<n; j++)
         matrix_x[i][j]-=t[i];
//--- syrk C=alpha*A^H*A+beta*C (beta=0 and not considered)
   matrix_c=matrix::Zeros(m,m);
   for(i=0; i<m; i++)
     {
      for(j=0; j<n; j++)
        {
         double v=matrix_x[i][j]/(n-1);
         for(int i_=i; i_<m; i_++)
            matrix_c[i][i_]+=v*matrix_x[i_][j];
        }
     }
//--- force symmetricity
   for(i=0; i<m-1; i++)
      for(j=i+1; j<m; j++)
         matrix_c[j][i]=matrix_c[i][j];
//---
   return(true);
  }
```

MQL5 example:

```
   matrix matrix_a={{3,-2.1},{1.1,-1},{0.12,4.3}};
   Print("covariation cols\n",matrix_a.Cov(false));
   Print("covariation rows\n",matrix_a.Cov());

   vector vector_a=matrix_a.Col(0);
   vector vector_b=matrix_a.Col(1);
   Print("covariation vectors\n",vector_a.Cov(vector_b));
```

```
  /*
   covariation cols
   [[2.144133333333333,-4.286]
    [-4.286,11.71]]
   covariation rows
   [[13.005,5.355,-10.659]
    [5.355,2.205,-4.389]
    [-10.659,-4.389,8.736199999999998]]
   covariation vectors
   [[2.144133333333333,-4.286]
    [-4.286,11.71]]
  */
```

Python example:

```
import numpy as np
matrix_a=np.array([[3,-2.1],[1.1,-1],[0.12,4.3]])
matrix_c=np.cov(matrix_a,rowvar=False)
print("covariation cols\n",matrix_c)
matrix_c2=np.cov(matrix_a)
print("covariation rows\n",matrix_c2)

vector_a=matrix_a[:,0]
vector_b=matrix_a[:,1]
matrix_c3=np.cov(vector_a,vector_b)
print("covariation vectors\n",matrix_c3)

covariation cols
 [[ 2.14413333 -4.286     ]
 [-4.286      11.71      ]]
covariation rows
 [[ 13.005     5.355   -10.659 ]
 [  5.355     2.205    -4.389 ]
 [-10.659    -4.389     8.7362]]
covariation vectors
 [[ 2.14413333 -4.286     ]
 [-4.286      11.71      ]]
```

Correlate

Compute the cross-correlation of two vectors.

```
vector vector::Correlate(
  const vector&           v,         // vector
  ENUM_VECTOR_CONVOLVE    mode       // mode
   );
```

Parameters

v

[in] Second vector.

mode

[in] The 'mode' parameter determines the linear convolution calculation mode. Value from the ENUM_VECTOR_CONVOLVE enumeration.

Return Value

Cross-correlation of two vectors.

Note

The 'mode' parameter determines the linear convolution calculation mode.

A simple algorithm for calculating the correlation coefficient of two vectors using MQL5:

```
vector VectorCrossCorrelationFull(const vector& a,const vector& b)
  {
   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=m+n-1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
      for(int i_=i; i_<i+m; i_++)
         c[i_]+=b[n-i-1]*a[i_-i];

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorCrossCorrelationSame(const vector& a,const vector& b)
  {
   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=MathMax(m,n);
   vector c=vector::Zeros(size);
   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-size/2+1;
         if(k>=0 && k<size)
            c[k]+=b[n-i-1]*a[i_-i];
        }
     }

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorCrossCorrelationValid(const vector& a,const vector& b)
  {
   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=MathMax(m,n)-MathMin(m,n)+1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-n+1;
         if(k>=0 && k<size)
            c[k]+=b[n-i-1]*a[i_-i];
        }
     }

   return(c);
  }
```

MQL5 example:

```
   vector a={1,2,3,4,5};
   vector b={0,1,0.5};

   Print("full\n",a.Correlate(b,VECTOR_CONVOLVE_FULL));
   Print("same\n",a.Correlate(b,VECTOR_CONVOLVE_SAME));
   Print("valid\n",a.Correlate(b,VECTOR_CONVOLVE_VALID));
   Print("full\n",b.Correlate(a,VECTOR_CONVOLVE_FULL));

  /*
   full
```

```
    [0.5,2,3.5,5,6.5,5,0]
    same
    [2,3.5,5,6.5,5]
    valid
    [3.5,5,6.5]
    full
    [0,5,6.5,5,3.5,2,0.5]
  */
```

Python example:

```
import numpy as np
a=[1,2,3,4,5]
b=[0,1,0.5]

print("full\n",np.correlate(a,b,'full'))
print("same\n",np.correlate(a,b,'same'));
print("valid\n",np.correlate(a,b,'valid'));
print("full\n",np.correlate(b,a,'full'))

full
 [0.5 2.  3.5 5.  6.5 5.  0. ]
same
 [2.  3.5 5.  6.5 5. ]
valid
 [3.5 5.  6.5]
full
 [0.  5.  6.5 5.  3.5 2.  0.5]
```

Convolve

Return the discrete, linear convolution of two vectors

```
vector vector::Convolve(
  const vector&           v,         // vector
  ENUM_VECTOR_CONVOLVE    mode       // mode
   );
```

Parameters

v

[out] Second vector.

mode

[in] The 'mode' parameter determines the linear convolution calculation mode ENUM_VECTOR_CONVOLVE.

Return Value

Discrete, linear convolution of two vectors.

A simple algorithm for calculating the convolution of two vectors in MQL5:

```
vector VectorConvolutionFull(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionFull(b,a));

   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int    size=m+n-1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
      for(int i_=i; i_<i+m; i_++)
         c[i_]+=b[i]*a[i_-i];

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorConvolutionSame(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionSame(b,a));

   int    m=(int)a.Size();
   int    n=(int)b.Size();
   int       size=MathMax(m,n);
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-size/2+1;
         if(k>=0 && k<size)
            c[k]+=b[i]*a[i_-i];
        }
     }

   return(c);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
vector VectorConvolutionValid(const vector& a,const vector& b)
  {
   if(a.Size()<b.Size())
      return(VectorConvolutionValid(b,a));

   int       m=(int)a.Size();
   int       n=(int)b.Size();
   int       size=MathMax(m,n)-MathMin(m,n)+1;
   vector c=vector::Zeros(size);

   for(int i=0; i<n; i++)
     {
      for(int i_=i; i_<i+m; i_++)
        {
         int k=i_-n+1;
         if(k>=0 && k<size)
            c[k]+=b[i]*a[i_-i];
        }
     }

   return(c);
  }
```

MQL5 example:

```
  vector a= {1, 2, 3, 4, 5};
  vector b= {0, 1, 0.5};

  Print("full\n", a.Convolve(b, VECTOR_CONVOLVE_FULL));
  Print("same\n", a.Convolve(b, VECTOR_CONVOLVE_SAME));
```

```
   Print("valid\n", a.Convolve(b, VECTOR_CONVOLVE_VALID));

   /*
    full
    [0,1,2.5,4,5.5,7,2.5]
    same
    [1,2.5,4,5.5,7]
    valid
    [2.5,4,5.5]
   */
```

Python example:

```
import numpy as np
a=[1,2,3,4,5]
b=[0,1,0.5]

print("full\n",np.convolve(a,b,'full'))
print("same\n",np.convolve(a,b,'same'));
print("valid\n",np.convolve(a,b,'valid'));


full
 [0.  1.  2.5 4.  5.5 7.  2.5]
same
 [1.  2.5 4.  5.5 7. ]
valid
 [2.5 4.  5.5]
```

Matrix transformations

Matrix decomposition can be used in the following cases:

- as an intermediate step when solving systems of linear equations
- for matrix inversion
- when calculating determinants
- when finding eigenvalues and eigenvectors of a matrix
- when computing analytic functions of matrices
- when using the least squares method
- in the numerical solution of differential equations

Different matrix decomposition types are used depending on the problem.

Function	Action
Cholesky	Computes the Cholesky decomposition
Eig	Computes the eigenvalues and right eigenvectors of a square matrix
EigVals	Computes the eigenvalues of a general matrix
LU	LU factorization of a matrix as the product of a lower triangular matrix and an upper triangular matrix
LUP	LUP factorization with partial pivoting, which refers to LU decomposition with row permutations only: PA=LU
QR	Compute the qr factorization of a matrix
SVD	Singular Value Decomposition

Cholesky

Compute the Cholesky decomposition.

```
bool matrix::Cholesky(
  matrix&   L       // matrix
   );
```

Parameters

L key

[out] Lower triangular matrix.

Return Value

Returns true on success, false otherwise.

Note

Return the Cholesky decomposition, L * L.H, of the square matrix a, where L is lower-triangular and .H is the conjugate transpose operator (which is the ordinary transpose if a is real-valued). a must be Hermitian (symmetric if real-valued) and positive-definite. No checking is performed to verify whether a is Hermitian or not. In addition, only the lower-triangular and diagonal elements of a are used. Only L is actually returned.

Example

```
  matrix matrix_a= {{5.7998084, -2.1825367}, {-2.1825367, 9.85910595}};
  matrix matrix_l;
  Print("matrix_a\n", matrix_a);

  matrix_a.Cholesky(matrix_l);
  Print("matrix_l\n", matrix_l);
  Print("check\n", matrix_l.MatMul(matrix_l.Transpose()));

  /*
  matrix_a
  [[5.7998084,-2.1825367]
   [-2.1825367,9.85910595]]
  matrix_l
  [[2.408279136645086,0]
   [-0.9062640068544704,3.006291985133859]]
  check
  [[5.7998084,-2.1825367]
   [-2.1825367,9.85910595]]
  */
```

Eig

Compute the eigenvalues and right eigenvectors of a square matrix.

```
bool matrix::Eig(
  matrix&   eigen_vectors,          // matrix of eigenvectors
  vector&   eigen_values            // vector of eigenvalues
   );
```

Complex solution of eigenvalues and eigenvectors

```
bool matrix::Eig(
  matrix<complex>& eigen_vectors, // matrix of eigenvectors
  vector<complex>& eigen_values   // vector of eigenvalues
   );
```

Parameters

eigen_vectors

[out] Matrix of vertical eigenvectors.

eigen_values

[out] Vector of eigenvalues.

Return Value

Returns true on success, false otherwise.

Note

If a complex solution is encountered when calculating eigenvalues, the calculation is stopped and the error code is set to 4019 (ERR_MATH_OVERFLOW). Use the complex overload of the Eig method to obtain a complete solution in complex space

If a complex eigenvalue has an imaginary part equal to zero, then it is a real eigenvalue. This can be seen in the example below.

Example

```
void OnStart()
  {
   matrix matrix_a =
     {
        {-3.474589, 1.106384, -9.091977,-3.925227 },
        {-5.522139, 2.366887,-15.162351,-6.357512 },
        { 8.394926,-2.960067, 22.292115, 9.524129 },
        { 7.803242,-2.080287, 19.217706, 8.186645 }
     };

   matrix eigen_vectors;
   vector eigen_values;

   bool res=matrix_a.Eig(eigen_vectors, eigen_values);
   Print("res=",res,"  error=",GetLastError());
   Print("Eigen vectors:\n",eigen_vectors, "\nEigen Values:\n",eigen_values);

//--- check correctness A * v = lambda * v
   int vectors=0;
   for(ulong n=0; n<eigen_values.Size(); n++)
     {
      vector eigen_vector=eigen_vectors.Col(n);
      vector vector_d1   =eigen_vector*eigen_values[n];
      vector vector_d2   =matrix_a.MatMul(eigen_vector);

      ulong errors=vector_d1.Compare(vector_d2,1e-13);
      if(errors==0)
         vectors++;
     }
   Print("vectors=",vectors);

//--- complex solution
   matrix<complex> eigen_vectors_c;
   vector<complex> eigen_values_c;
   ResetLastError();
   res=matrix_a.Eig(eigen_vectors_c,eigen_values_c);
   Print("res=",res,"  error=",GetLastError());
   Print("Eigen vectors:\n",eigen_vectors_c, "\nEigen Values:\n",eigen_values_c);

//--- check correctness A * v = lambda * v
   matrixc matrix_c;
   matrix_c.Assign(matrix_a);
   vectors=0;
   for(ulong n=0; n<eigen_values_c.Size(); n++)
     {
      vectorc eigen_vector_c=eigen_vectors_c.Col(n);
      vectorc vector_c1     =eigen_vector_c*eigen_values_c[n];
      vectorc vector_c2     =matrix_c.MatMul(eigen_vector_c);

      ulong errors=vector_c1.Compare(vector_c2,1e-13);
      if(errors==0)
         vectors++;
     }
   Print("vectors=",vectors);
  }
/* Result
   res=true  error=4019
   Eigen vectors:
   [[0.2649667608713664]
    [0.4488818803991876]
    [-0.6527335897527492]
    [-0.5497604331807768]]
   Eigen Values:
    [28.94158645962942]
    vectors=1
    res=true  error=0
    Eigen vectors:
    [[(0.2649667608713664,0),(0.2227392219204762,0.3745470492013296),(0.40328543908977
     [(0.4488818803991876,0),(-0.2613452530342438,-0.3685707727819327),(-0.39685061263
     [(-0.6527335897527492,0),(-0.3418479634708521,-0.3299830378162041),(-0.3553021031
     [(-0.5497604331807768,0),(0.523227993452828,0.3262508584080381),(0.35128355961436
    Eigen Values:
    [(28.94158645962942,0),(0.01022501104810897,0.02954980822331488),(0.01022501104810
    vectors=3
*/
```

EigVals

Compute the eigenvalues of a general matrix.

```
bool matrix::EigVals(
  vector&   eigen_values      // vector of eigenvalues
   );
```

Parameters

eigen_values

[out] Vector of right eigenvalues.

Return Value

Returns true on success, false otherwise.

Note

The only difference between EigVals and Eig is that EigVals does not calculate eigenvectors, while it only calculates eigenvalues.

LU

LU factorization of a matrix as the product of a lower triangular matrix and an upper triangular matrix.

```
bool matrix::LU(
   matrix&   L,      // lower triangular matrix
   matrix&   U       // upper triangular matrix
    );
```

Parameters

L key

[out] Lower triangular matrix.

U

[out] Upper triangular matrix.

Return Value

Returns true on success, false otherwise.

Example

```
   matrix matrix_a={{1,2,3,4},
                    {5,2,6,7},
                    {8,9,3,10},
                    {11,12,14,4}};
   matrix matrix_l,matrix_u;
//--- LU decomposition
   matrix_a.LU(matrix_l,matrix_u);
   Print("matrix_l\n",matrix_l);
   Print("matrix_u\n",matrix_u);
//--- check if A = L * U
   Print("check\n",matrix_l.MatMul(matrix_u));


   /*
   matrix_l
   [[1,0,0,0]
    [5,1,0,0]
    [8,0.875,1,0]
    [11,1.25,0.5904761904761905,1]]
   matrix_u
   [[1,2,3,4]
    [0,-8,-9,-13]
    [0,0,-13.125,-10.625]
    [0,0,0,-17.47619047619047]]
   check
   [[1,2,3,4]
```

```
     [5,2,6,7]
     [8,9,3,10]
     [11,12,14,4]]
    */
```

LUP

LUP factorization with partial pivoting, which refers to LU decomposition with row permutations only: PA=LU.

```
bool  LUP(
   matrix&   L,      // lower triangular matrix
   matrix&   U,      // upper triangular matrix
   matrix&   P       // permutations matrix
    );
```

Parameters

L key

[out] Lower triangular matrix.

U

[out] Upper triangular matrix.

P

[out] Permutations matrix

Return Value

Returns true on success, false otherwise.

Example

```
   matrix matrix_a={{1,2,3,4},
                    {5,2,6,7},
                    {8,9,3,10},
                    {11,12,14,4}};
   matrix matrix_l,matrix_u,matrix_p;
//--- LUP decomposition
   matrix_a.LUP(matrix_l,matrix_u,matrix_p);
   Print("matrix_l\n",matrix_l);
   Print("matrix_u\n",matrix_u);
   Print("matrix_p\n",matrix_p);
//--- check if P * A = L * U
   Print("P * A\n",matrix_p.MatMul(matrix_a));
   Print("L * U\n",matrix_l.MatMul(matrix_u));

   /*
   matrix_l
   [[1,0,0,0]
    [0.4545454545454545,1,0,0]
    [0.7272727272727273,-0.07894736842105282,1,0]
    [0.09090909090909091,-0.2631578947368421,-0.2262773722627738,1]]
   matrix_u
   [[11,12,14,4]
```

```
  [0,-3.454545454545454,-0.3636363636363633,5.181818181818182]
  [0,0,-7.210526315789473,7.500000000000001]
  [0,0,0,6.697080291970803]]
 matrix_p
 [[0,0,0,1]
  [0,1,0,0]
  [0,0,1,0]
  [1,0,0,0]]
 P * A
 [[11,12,14,4]
  [5,2,6,7]
  [8,9,3,10]
  [1,2,3,4]]
 L * U
 [[11,12,14,4]
  [5,2,6,7]
  [8,9,3.000000000000001,10]
  [1,2,3,4]]
 */
```

QR

Compute the qr factorization of a matrix.

```
bool  QR(
  matrix&  Q,      // matrix with orthonormal columns
  matrix&  R       // upper-triangular matrix
   );
```

Parameters

Q

[out] A matrix with orthonormal columns. When mode = 'complete' the result is an orthogonal/unitary matrix depending on whether or not a is real/complex. The determinant may be either +/- 1 in that case. In case the number of dimensions in the input array is greater than 2 then a stack of the matrices with above properties is returned.

R key

[out] Upper triangular matrix.

Return Value

Returns true on success, false otherwise.

Example

```
//---  A*x = b
  matrix A = {{0, 1}, {1, 1}, {1, 1}, {2, 1}};
  Print("A \n", A);
  vector b = {1, 2, 2, 3};
  Print("b \n", b);
//--- A = Q*R
  matrix q, r;
  A.QR(q, r);
  Print("q \n", q);
  Print("r \n", r);
  matrix qr=q.MatMul(r);
  Print("qr \n", qr);
  /*
  A
  [[0,1]
  [1,1]
  [1,1]
  [2,1]]
  b
  [1,2,2,3]
  q
  [[0.4082482904638631,-0.8164965809277259,-1.110223024625157e-16,-0.4082482904638631]
  [0.4625425214347352,-0.03745747856526496,0.7041241452319315,0.5374574785652647]
  [-0.5374574785652648,-0.03745747856526496,0.7041241452319316,-0.4625425214347352]
[-0.5749149571305296,-0.5749149571305299,-0.09175170953613698,0.5749149571305296]]
r
[[-1.224744871391589,-0.2415816237971962]
[-1.22474487139159,-1.466326495188786]
[1.224744871391589,1.316496580927726]
[1.224744871391589,0.2415816237971961]]
qr
[[-1.110223024625157e-16,1]
[1,0.9999999999999999]
[1,1]
[2,1]]
*/
```

SVD

Singular Value Decomposition.

```
bool matrix::SVD(
  matrix&   U,                     // unitary matrix
  matrix&   V,                     // unitary matrix
  vector&   singular_values        // singular values vector
   );
```

Parameters

U

[out] Unitary matrix of order m, consisting of left singular vectors.

V

[out] Unitary matrix of order n, consisting of right singular vectors.

singular_values

[out] Singular values

Return Value

Returns true on success, false otherwise.

Example

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a=a-4;
  Print("matrix a \n", a);
  a.Reshape(3, 3);
  matrix b=a;
  Print("matrix b \n", b);
//--- execute SVD decomposition
  matrix U, V;
  vector singular_values;
  b.SVD(U, V, singular_values);
  Print("U \n", U);
  Print("V \n", V);
  Print("singular_values = ", singular_values);

// check block
//--- U * singular diagonal * V = A
  matrix matrix_s;
  matrix_s.Diag(singular_values);
  Print("matrix_s \n", matrix_s);
  matrix matrix_vt=V.Transpose();
  Print("matrix_vt \n", matrix_vt);
  matrix matrix_usvt=(U.MatMul(matrix_s)).MatMul(matrix_vt);
  Print("matrix_usvt \n", matrix_usvt);
  ulong errors=(int)b.Compare(matrix_usvt, 1e-9);
  double res=(errors==0);
  Print("errors=", errors);

//---- another check
  matrix U_Ut=U.MatMul(U.Transpose());
  Print("U_Ut \n", U_Ut);
  Print("Ut_U \n", (U.Transpose()).MatMul(U));

  matrix vt_V=matrix_vt.MatMul(V);
  Print("vt_V \n", vt_V);
  Print("V_vt \n", V.MatMul(matrix_vt));

  /*
  matrix a
  [[-4,-3,-2,-1,0,1,2,3,4]]
  matrix b
  [[-4,-3,-2]
   [-1,0,1]
   [2,3,4]]
  U
  [[-0.7071067811865474,0.5773502691896254,0.408248290463863]
   [-6.827109697437648e-17,0.5773502691896253,-0.8164965809277256]
   [0.7071067811865472,0.5773502691896255,0.4082482904638627]]
  V
  [[0.5773502691896258,-0.7071067811865474,-0.408248290463863]
   [0.5773502691896258,1.779939029415334e-16,0.8164965809277258]
   [0.5773502691896256,0.7071067811865474,-0.408248290463863]]
  singular_values = [7.348469228349533,2.449489742783175,3.277709923350408e-17]

  matrix_s
  [[7.348469228349533,0,0]
   [0,2.449489742783175,0]
   [0,0,3.277709923350408e-17]]
  matrix_vt
  [[0.5773502691896258,0.5773502691896258,0.5773502691896256]
   [-0.7071067811865474,1.779939029415334e-16,0.7071067811865474]
   [-0.408248290463863,0.8164965809277258,-0.408248290463863]]
  matrix_usvt
  [[-3.999999999999997,-2.999999999999999,-2]
   [-0.9999999999999981,-5.977974170712231e-17,0.9999999999999974]
   [2,2.999999999999999,3.999999999999996]]
  errors=0

  U_Ut
  [[0.9999999999999993,-1.665334536937735e-16,-1.665334536937735e-16]
   [-1.665334536937735e-16,0.9999999999999987,-5.551115123125783e-17]
   [-1.665334536937735e-16,-5.551115123125783e-17,0.999999999999999]]
   Ut_U
   [[0.9999999999999993,-5.551115123125783e-17,-1.110223024625157e-16]
    [-5.551115123125783e-17,0.9999999999999987,2.498001805406602e-16]
    [-1.110223024625157e-16,2.498001805406602e-16,0.999999999999999]]
   vt_V
   [[1,-5.551115123125783e-17,0]
    [-5.551115123125783e-17,0.9999999999999996,1.110223024625157e-16]
    [0,1.110223024625157e-16,0.9999999999999996]]
   V_vt
   [[0.9999999999999999,1.110223024625157e-16,1.942890293094024e-16]
    [1.110223024625157e-16,0.9999999999999998,1.665334536937735e-16]
    [1.942890293094024e-16,1.665334536937735e-16,0.9999999999999996]
   */
  }
```

Statistical methods

Methods for computing descriptive statistics of matrices and vectors.

Function	Action
ArgMax	Return the index of the maximum value
ArgMin	Return the index of the minimum value
Max	Return the maximum value in a matrix/vector
Min	Return the minimum value in a matrix/vector
Ptp	Return the range of values of a matrix/vector or of the given matrix axis
Sum	Return the sum of the matrix/vector elements which can also be performed for the given axis (axes)
Prod	Return the product of the matrix/vector elements which can also be performed for the given axis
CumSum	Return the cumulative sum of matrix/vector elements, including those along the given axis
CumProd	Return the cumulative product of matrix/vector elements, including those along the given axis
Percentile	Return the specified percentile of values of matrix/vector elements or elements along the specified axis
Quantile	Return the specified quantile of values of matrix/vector elements or elements along the specified axis
Median	Compute the median of the matrix/vector elements
Mean	Compute the arithmetic mean of element values
Average	Compute the weighted mean of matrix/vector values
Std	Return the standard deviation of values of matrix/vector elements or of elements along the given axis
Var	Compute the variance of values of matrix/vector elements
LinearRegression	Calculate a vector/matrix with calculated linear regression values

ArgMax

Return the index of the maximum value.

```
ulong vector::ArgMax();

ulong matrix::ArgMax();

vector matrix::ArgMax(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Index of the maximum value.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_max=matrix_a.ArgMax(0);
   vector rows_max=matrix_a.ArgMax(1);
   ulong  matrix_max=matrix_a.ArgMax();

   Print("cols_max=",cols_max);
   Print("rows_max=",rows_max);
   Print("max index ",matrix_max,"  max value ",matrix_a.Flat(matrix_max));

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_max=[0,3,1]
   rows_max=[0,2,0,1]
   max index 5  max value 12.0
   */
```

ArgMin

Return the index of the minimum value.

```
ulong vector::ArgMin();

ulong matrix::ArgMin();

vector matrix::ArgMin(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Index of the minimum value.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_min=matrix_a.ArgMin(0);
   vector rows_min=matrix_a.ArgMin(1);
   ulong  matrix_min=matrix_a.ArgMin();

   Print("cols_min=",cols_min);
   Print("rows_min=",rows_min);
   Print("min index ",matrix_min,"  min value ",matrix_a.Flat(matrix_min));

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_min=[1,0,0]
   rows_min=[2,0,2,0]
   min index 3  min value 1.0
   */
```

Max

Return the maximum value in a matrix/vector.

```
double vector::Max();

double matrix::Max();

vector matrix::Max(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Maximum value in a matrix/vector.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_max=matrix_a.Max(0);
   vector rows_max=matrix_a.Max(1);
   double matrix_max=matrix_a.Max();

   Print("cols_max=",cols_max);
   Print("rows_max=",rows_max);
   Print("max value ",matrix_max);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_max=[10,11,12]
   rows_max=[10,12,6,11]
   max value 12.0
   */
```

Min

Return the minimum value in a matrix/vector.

```
double vector::Min();

double matrix::Min();

vector matrix::Min(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Minimum value in a matrix/vector.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_min=matrix_a.Min(0);
   vector rows_min=matrix_a.Min(1);
   double matrix_min=matrix_a.Min();

   Print("cols_min=",cols_min);
   Print("rows_min=",rows_min);
   Print("min value ",matrix_min);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_min=[1,3,2]
   rows_min=[2,1,4,7]
   min value 1.0
   */
```

Ptp

Return the range of values of a matrix/vector or of the given matrix axis, equivalent to Max() - Min(). Ptp - Peak to peak.

```
double vector::Ptp();

double matrix::Ptp();

vector matrix::Ptp(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Vector with ranges of values (maximum - minimum) .

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_ptp=matrix_a.Ptp(0);
   vector rows_ptp=matrix_a.Ptp(1);
   double matrix_ptp=matrix_a.Ptp();

   Print("cols_ptp  ",cols_ptp);
   Print("rows_ptp  ",rows_ptp);
   Print("ptp value  ",matrix_ptp);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_ptp [9,8,10]
   rows_ptp [8,11,2,4]
   ptp value  11.0
   */
```

Sum

Return the sum of the matrix/vector elements which can also be performed for the given axis (axes).

```
double vector::Sum();

double matrix::Sum();

vector matrix::Sum(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Sum of the matrix/vector elements which can also be performed for the given axis (axes).

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_sum=matrix_a.Sum(0);
   vector rows_sum=matrix_a.Sum(1);
   double matrix_sum=matrix_a.Sum();

   Print("cols_sum=",cols_sum);
   Print("rows_sum=",rows_sum);
   Print("sum value ",matrix_sum);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_sum=[24,27,27]
   rows_sum=[15,21,15,27]
   sum value 78.0
   */
```

Prod

Return the product of the matrix/vector elements which can also be performed for the given axis.

```
double vector::Prod(
  const double  initial=1       // initial multiplier
   );

double matrix::Prod(
  const double  initial=1       // initial multiplier
   );

vector matrix::Prod(
  const int     axis,           // axis
  const double  initial=1       // initial multiplier
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

initial=1

[in] Initial multiplier.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vector cols_prod=matrix_a.Prod(0);
   vector rows_prod=matrix_a.Prod(1);
   double matrix_prod=matrix_a.Prod();

   Print("cols_prod=",cols_prod);
   cols_prod=matrix_a.Prod(0,0.1);
   Print("cols_prod=",cols_prod);
   Print("rows_prod=",rows_prod);
   Print("prod value ",matrix_prod);

  /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_prod=[420,1320,864]
   cols_prod=[42,132,86.40000000000001]
   rows_prod=[60,96,120,693]
  prod value 479001600.0
 */
```

CumSum

Return the cumulative sum of matrix/vector elements, including those along the given axis.

```
vector vector::CumSum();

vector matrix::CumSum();

matrix matrix::CumSum(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Cumulative sum of the elements along the given axis.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   matrix cols_cumsum=matrix_a.CumSum(0);
   matrix rows_cumsum=matrix_a.CumSum(1);
   vector cumsum_values=matrix_a.CumSum();

   Print("cols_cumsum\n",cols_cumsum);
   Print("rows_cumsum\n",rows_cumsum);
   Print("cumsum values  ",cumsum_values);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_cumsum
   [[10,3,2]
    [11,11,14]
    [17,16,18]
    [24,27,27]]
   rows_cumsum
   [[10,13,15]
    [1,9,21]
    [6,11,15]
```

```
     [7,18,27]]
    cumsum values  [10,13,15,16,24,36,42,47,51,58,69,78]
    */
```

CumProd

Return the cumulative product of matrix/vector elements, including those along the given axis.

```
vector vector::CumProd();

vector matrix::CumProd();

matrix matrix::CumProd(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis for each column (i.e., over the rows), 1 — vertical axis for each row (i.e. over the columns)

Return Value

Cumulative product of the elements along the given axis.

Example

```
   matrix matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   matrix cols_cumprod=matrix_a.CumProd(0);
   matrix rows_cumprod=matrix_a.CumProd(1);
   vector cumprod_values=matrix_a.CumProd();

   Print("cols_cumprod\n",cols_cumprod);
   Print("rows_cumprod\n",rows_cumprod);
   Print("cumprod values  ",cumprod_values);

   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_cumprod
   [[10,3,2]
    [10,24,24]
    [60,120,96]
    [420,1320,864]]
   rows_cumprod
   [[10,30,60]
    [1,8,96]
```

```
  [6,30,120]
  [7,77,693]]
cumprod values  [10,30,60,60,480,5760,34560,172800,691200,4838400,53222400,47900160
*/
```

Percentile

Return the specified percentile of values of matrix/vector elements or elements along the specified axis.

```
double vector::Percentile(
   const int   percent      //
    );

double matrix::Percentile(
   const int   percent      //
    );

vector matrix::Percentile(
   const int   percent,     //
   const int   axis         // axis
    );
```

Parameters

percent

[in] Percentiles to compute, which must be between 0 and 100 inclusive.

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Percentile: scalar or vector.

Note

Valid values of the 'percent' parameter are in the range [0, 100]. A linear algorithm is used to calculate percentiles. The correct calculation of percentiles requires the sequence to be sorted.

Example

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_percentile=matrix_a.Percentile(50,0);
   vectorf rows_percentile=matrix_a.Percentile(50,1);
   float matrix_percentile=matrix_a.Percentile(50);

   Print("cols_percentile ",cols_percentile);
   Print("rows_percentile ",rows_percentile);
   Print("percentile value  ",matrix_percentile);


   /*
```

```
matrix_a
[[1,2,3]
 [4,5,6]
 [7,8,9]
 [10,11,12]]
cols_percentile [5.5,6.5,7.5]
rows_percentile [2,5,8,11]
percentile value  6.5
*/
```

Quantile

Return the specified quantile of values of matrix/vector elements or elements along the specified axis.

```
double vector::Quantile(
   const double quantile      // quantile
   );

double matrix::Quantile(
   const double quantile      // quantile
   );

vector matrix::Quantile(
   const double quantile,     // quantile
   const int    axis          // axis
   );
```

Parameters

quantile

[in] Quantile to compute, which must be between 0 and 1 inclusive.

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Quantile: scalar or vector.

Note

The 'quantile' parameter takes values in the range [0, 1]. A linear algorithm is used to calculate quantiles. The correct calculation of quantiles requires the sequence to be sorted.

Example

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_quantile=matrix_a.Quantile(0.5,0);
   vectorf rows_quantile=matrix_a.Quantile(0.5,1);
   float matrix_quantile=matrix_a.Quantile(0.5);

   Print("cols_quantile ",cols_quantile);
   Print("rows_quantile ",rows_quantile);
   Print("quantile value  ",matrix_quantile);

   /*
   matrix_a
```

```
[[1,2,3]
 [4,5,6]
 [7,8,9]
 [10,11,12]]
cols_quantile [5.5,6.5,7.5]
rows_quantile [2,5,8,11]
quantile value  6.5
*/
```

Median

Compute the median of the matrix/vector elements.

```
double vector::Median();

double matrix::Median();

vector matrix::Median(
  const int  axis       // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Median: scalar or vector.

Note

The median is the middle value that separates the highest half of the array/vector elements from the lowest half of elements. Same as Quantile(0.5) and Percentile(50). The correct calculation of median requires the sequence to be sorted.

Example

```
   matrixf matrix_a={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_median=matrix_a.Median(0);
   vectorf rows_median=matrix_a.Median(1);
   float matrix_median=matrix_a.Median();

   Print("cols_median ",cols_median);
   Print("rows_median ",rows_median);
   Print("median value  ",matrix_median);


   /*
   matrix_a
   [[1,2,3]
    [4,5,6]
    [7,8,9]
    [10,11,12]]
   cols_median [5.5,6.5,7.5]
```

```
rows_median [2,5,8,11]
median value  6.5
*/
```

Mean

Compute the arithmetic mean of element values.

```
double vector::Mean();

double matrix::Mean();

vector matrix::Mean(
  const int  axis      // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Arithmetic mean of element values.

Example

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_mean=matrix_a.Mean(0);
   vectorf rows_mean=matrix_a.Mean(1);
   float matrix_mean=matrix_a.Mean();

   Print("cols_mean ",cols_mean);
   Print("rows_mean ",rows_mean);
   Print("mean value  ",matrix_mean);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_mean [6,6.75,6.75]
   rows_mean [5,7,5,9]
   mean value  6.5
   */
```

Average

Compute the weighted mean of matrix/vector values.

```
double vector::Average(
  const vector& weigts       // weights vector
   );

double matrix::Average(
  const matrix& weigts       // weights matrix
);

vector matrix::Average(
  const matrix& weigts,      // weights matrix
  const int     axis         // axis
   );
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

Weighted mean: scalar or vector.

Note

The weight matrix/vector is associated with the main matrix/vector.

Example

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   matrixf matrix_w=matrixf::Ones(4,3);
   Print("matrix_a\n",matrix_a);

   vectorf cols_average=matrix_a.Average(matrix_w,0);
   vectorf rows_average=matrix_a.Average(matrix_w,1);
   float matrix_average=matrix_a.Average(matrix_w);

   Print("cols_average ",cols_average);
   Print("rows_average ",rows_average);
   Print("average value  ",matrix_average);


   /*
   matrix_a
   [[10,3,2]
```

```
     [1,8,12]
     [6,5,4]
     [7,11,9]]
    cols_average [6,6.75,6.75]
    rows_average [5,7,5,9]
    average value  6.5
    */ value  6.5
```

Std

Return the standard deviation of values of matrix/vector elements or of elements along the given axis.

```
double vector::Std();

double vector::Std(
   const int   ddof       // delta degrees of freedom
);

double matrix::Std();

vector matrix::Std(
   const int   axis       // axis
);

vector matrix::Std(
   const int   axis,      // axis
   const int   ddof       // delta degrees of freedom
);
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

ddof

[in] "Delta Degrees of Freedom": the divisor used in the calculation is N - ddof, where N represents the number of elements. By default ddof is zero.

Return Value

Standard_deviation: scalar or vector.

Note

The standard deviation is the square root of the average of the squared deviations from the mean, i. e., std = sqrt(mean(x)), where x = abs(a - a.mean())**2.

The average squared deviation is typically calculated as x.sum() / (N - ddof), where N = len(x).

Expression with ddof=0 sometimes called the "population standard deviation". If ddof>0 (most commonly used 1), the resulting quantity is sometimes called the "sample standard deviation".

Example

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_std=matrix_a.Std(0);
   vectorf rows_std=matrix_a.Std(1);
   float matrix_std=matrix_a.Std();

   Print("cols_std ",cols_std);
   Print("rows_std ",rows_std);
   Print("std value  ",matrix_std);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_std [3.2403703,3.0310888,3.9607449]
   rows_std [3.5590262,4.5460606,0.81649661,1.6329932]
   std value  3.452052593231201
   */
```

Var

Compute the variance of values of matrix/vector elements.

```
double vector::Var();

double vector::Var(
  const int  ddof      // delta degrees of freedom
);

double matrix::Var();

vector matrix::Var(
  const int  axis      // axis
);

vector matrix::Var(
  const int  axis,     // axis
  const int  ddof      // delta degrees of freedom
);
```

Parameters

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

ddof

[in] “Delta Degrees of Freedom”: the divisor used in the calculation is N - ddof, where N represents the number of elements. By default ddof is zero.

Return Value

Variance: scalar or vector.

Note

The variance is the average of the squared deviations from the mean, i.e., var = mean(x), where x = abs(a - a.mean())**2.

The mean is typically calculated as x.sum() / (N - ddof), where N = len(x).

Expression with ddof=0 sometimes called the “population variance”. If ddof>0 (most commonly used 1), the resulting quantity is sometimes called the “sample variance”.

Example

```
   matrixf matrix_a={{10,3,2},{1,8,12},{6,5,4},{7,11,9}};
   Print("matrix_a\n",matrix_a);

   vectorf cols_var=matrix_a.Var(0);
   vectorf rows_var=matrix_a.Var(1);
   float matrix_var=matrix_a.Var();

   Print("cols_var ",cols_var);
   Print("rows_var ",rows_var);
   Print("var value  ",matrix_var);


   /*
   matrix_a
   [[10,3,2]
    [1,8,12]
    [6,5,4]
    [7,11,9]]
   cols_var [10.5,9.1875,15.6875]
   rows_var [12.666667,20.666666,0.66666669,2.6666667]
   var value  11.916666984558105
   */
```

LinearRegression

Calculate a vector/matrix with calculated linear regression values.

```
vector vector::LinearRegression();

matrix matrix::LinearRegression(
  ENUM_MATRIX_AXIS  axis=AXIS_NONE      // axis along which regression is calculated
   );
```

Parameters

axis

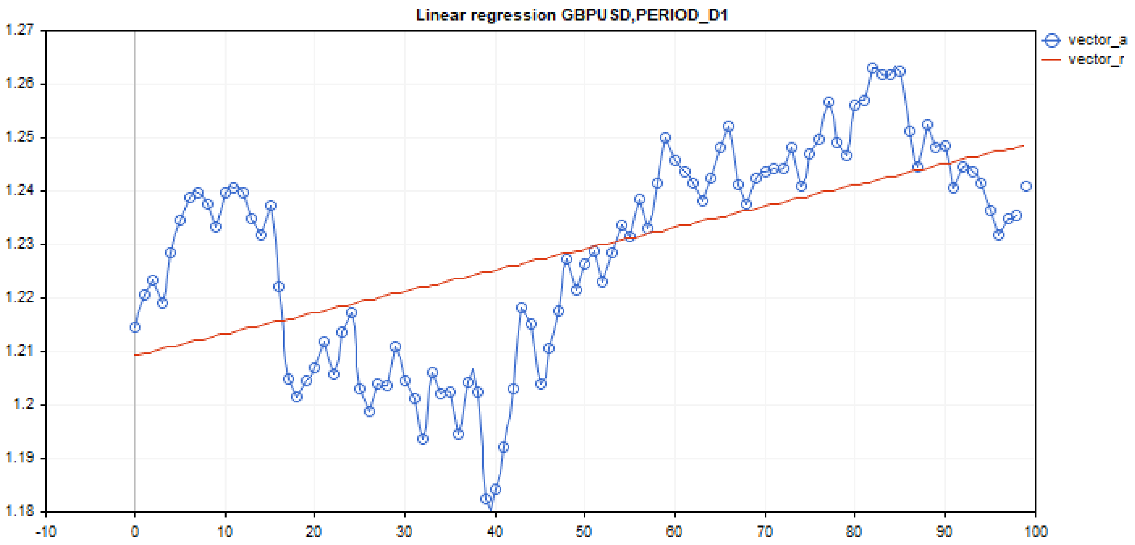
[in] Specifying the axis along which the regression is calculated. ENUM_MATRIX_AXIS enumeration value (AXIS_HORZ — horizontal axis, AXIS_VERT — vertical axis).

Return Value

Vector or matrix with calculated linear regression values.

Note

Linear regression is calculated using the standard regression equation: y (x) = a * x + b, where a is the line slope, while b is its Y axis shift.



Example:

```
#include <Graphics\Graphic.mqh>

#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector vector_a;
   vector_a.CopyRates(_Symbol,_Period,COPY_RATES_CLOSE,1,100);
   vector vector_r=vector_a.LinearRegression();

//--- switch off chart show
   ChartSetInteger(0,CHART_SHOW,false);

//--- arrays for drawing a graph
   double x[];
   double y1[];
   double y2[];
   ArrayResize(x,uint(vector_a.Size()));
   ArrayResize(y1,uint(vector_a.Size()));
   ArrayResize(y2,uint(vector_a.Size()));
   for(ulong i=0; i<vector_a.Size(); i++)
     {
      x[i]=(double)i;
      y1[i]=vector_a[i];
      y2[i]=vector_r[i];
     }

//--- graph title
   string title="Linear regression "+_Symbol+","+EnumToString(_Period);

   long   chart=0;
   string name="LinearRegression";

//--- create graph
   CGraphic graphic;
   graphic.Create(chart,name,0,0,0,GRAPH_WIDTH,GRAPH_HEIGHT);
   graphic.BackgroundMain(title);
   graphic.BackgroundMainSize(12);

//--- activation function graph
   CCurve *curvef=graphic.CurveAdd(x,y1,CURVE_POINTS_AND_LINES);
   curvef.Name("vector_a");
   curvef.LinesWidth(2);
   curvef.LinesSmooth(true);
   curvef.LinesSmoothTension(1);
   curvef.LinesSmoothStep(10);

//--- derivatives of activation function
   CCurve *curved=graphic.CurveAdd(x,y2,CURVE_LINES);
   curved.Name("vector_r");
   curved.LinesWidth(2);
   curved.LinesSmooth(true);
   curved.LinesSmoothTension(1);
   curved.LinesSmoothStep(10);
   graphic.CurvePlotAll();
   graphic.Update();

//--- endless loop to recognize pressed keyboard buttons
   while(!IsStopped())
     {
      //--- press escape button to quit program
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
         break;
      //--- press PdDn to save graph picture
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0)
        {
         string file_names[];
         if(FileSelectDialog("Save Picture",NULL,"All files (*.*)|*.*",FSD_WRITE_FILE,
            continue;
         ChartScreenShot(0,file_names[0],GRAPH_WIDTH,GRAPH_HEIGHT);
        }
      Sleep(10);
     }

//--- clean up
   graphic.Destroy();
   ObjectDelete(chart,name);
   ChartSetInteger(0,CHART_SHOW,true);
  }
```

ENUM_MATRIX_AXIS

Enumeration for specifying the axis in all statistical functions for matrices.

ID	Description
AXIS_NONE	The axis is not specified. Calculation is performed over all matrix elements, as if it were a vector (see the Flat method).
AXIS_HORZ	Horizontal axis
AXIS_VERT	Vertical axis

Feature methods

These methods enable the receiving of matrix features, such as:

- number of rows
- number of columns
- norm
- condition number
- determinant
- matrix rank
- trace
- spectrum

Function	Action
Rows	Return the number of rows in a matrix
Cols	Return the number of columns in a matrix
Size	Return the size of vector
Norm	Return matrix or vector norm
Cond	Compute the condition number of a matrix
Det	Compute the determinant of a square invertible matrix
SLogDet	Compute the sign and logarithm of the determinant of an matrix
Rank	Return matrix rank using the Gaussian method
Trace	Return the sum along diagonals of the matrix
Spectrum	Compute spectrum of a matrix as the set of its eigenvalues from the product AT*A

Rows

Return the number of rows in a matrix.

```
ulong matrix::Rows()
```

Return Value

Integer.

Example

```
matrix m= {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}};
m.Reshape(3, 4);
Print("matrix m \n" , m);
Print("m.Rows()=", m.Rows());
Print("m.Cols()=", m.Cols());

/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 m.Rows()=3
 m.Cols()=4
*/
```

Cols

Return the number of columns in a matrix.

```
ulong matrix::Cols()
```

Return Value

Integer.

Example

```
matrix m= {{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}};
m.Reshape(3, 4);
Print("matrix m \n" , m);
Print("m.Cols()=", m.Cols());
Print("m.Rows()=", m.Rows());

/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 m.Cols()=4
 m.Rows()=3
*/
```

Size

Return the size of vector.

```
ulong vector::Size()
```

Return Value

Integer.

Example

```
matrix m={{1,2,3,4,5,6,7,8,9,10,11,12}};
m.Reshape(3,4);
Print("matrix m\n",m);
vector v=m.Row(1);
Print("v.Size()=",v.Size());
Print("v=",v);


/*
 matrix m
 [[1,2,3,4]
  [5,6,7,8]
  [9,10,11,12]]
 v.Size()=4
 v=[5,6,7,8]
*/
```

Norm

Return matrix or vector norm.

```
double vector::Norm(
  const ENUM_VECTOR_NORM  norm,      // vector norm
  const int              norm_p=2  // number of p-norm in case of VECTOR_NORM_P
   );

double matrix::Norm(
  const ENUM_MATRIX_NORM  norm      // matrix norm
   );
```

Parameters

norm

[in] Norm order

Return Value

Matrix or vector norm.

Note

- VECTOR_NORM_INF is the maximum absolute value among vector elements.
- VECTOR_NORM_MINUS_INF is the minimum absolute value of a vector.
- VECTOR_NORM_P is the P-norm of the vector. If norm_p=0, then this is the number of non-zero vector elements. norm_p=1 is the sum of absolute values of vector elements. norm_p=2 is the square root of the sum of squares of vector element values. The value of the norm_p parameter can be negative.
- MATRIX_NORM_FROBENIUS is the square root of the sum of the squares of the matrix element values. The Frobenius norm and the vector P2-norm are consistent.
- MATRIX_NORM_SPECTRAL is the maximum value of the matrix spectrum.
- MATRIX_NORM_NUCLEAR is the sum of the singular values of the matrix.
- MATRIX_NORM_INF is the maximum vector p1-norm among the vertical vectors of the matrix. The matrix inf-norm and the vector inf-norm are consistent.
- MATRIX_NORM_MINUS_INF is the minimum vector p1-norm among the vertical vectors of the matrix.
- MATRIX_NORM_P1 is the maximum vector p1-norm among horizontal matrix vectors.
- MATRIX_NORM_MINUS_P1 is the minimum vector p1-norm among horizontal matrix vectors.
- MATRIX_NORM_P2 is the highest singular value of the matrix.
- MATRIX_NORM_MINUS_P2 is the lowest singular value of a matrix.

A simple algorithm for calculating the P-norm of a vector in MQL5:

```
double VectorNormP(const vector& v,int norm_value)
  {
   ulong  i;
   double norm=0.0;
//---
   switch(norm_value)
     {
      case 0 :
         for(i=0; i<v.Size(); i++)
            if(v[i]!=0)
               norm+=1.0;
         break;
      case 1 :
         for(i=0; i<v.Size(); i++)
            norm+=MathAbs(v[i]);
         break;
      case 2 :
         for(i=0; i<v.Size(); i++)
            norm+=v[i]*v[i];
         norm=MathSqrt(norm);
         break;
      default :
         for(i=0; i<v.Size(); i++)
            norm+=MathPow(MathAbs(v[i]),norm_value);
         norm=MathPow(norm,1.0/norm_value);
     }
//---
   return(norm);
  }
```

MQL5 example:

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a=a-4;
  Print("matrix a \n", a);
  a.Reshape(3, 3);
  matrix b=a;
  Print("matrix b \n", b);
  Print("b.Norm(MATRIX_NORM_P2)=", b.Norm(MATRIX_NORM_FROBENIUS));
  Print("b.Norm(MATRIX_NORM_FROBENIUS)=", b.Norm(MATRIX_NORM_FROBENIUS));
  Print("b.Norm(MATRIX_NORM_INF)", b.Norm(MATRIX_NORM_INF));
  Print("b.Norm(MATRIX_NORM_MINUS_INF)", b.Norm(MATRIX_NORM_MINUS_INF));
  Print("b.Norm(MATRIX_NORM_P1)=)", b.Norm(MATRIX_NORM_P1));
  Print("b.Norm(MATRIX_NORM_MINUS_P1)=", b.Norm(MATRIX_NORM_MINUS_P1));
  Print("b.Norm(MATRIX_NORM_P2)=", b.Norm(MATRIX_NORM_P2));
  Print("b.Norm(MATRIX_NORM_MINUS_P2)=", b.Norm(MATRIX_NORM_MINUS_P2));

  /*
  matrix a
  [[-4,-3,-2,-1,0,1,2,3,4]]
  matrix b
  [[-4,-3,-2]
```

```
 [-1,0,1]
 [2,3,4]]
b.Norm(MATRIX_NORM_P2)=7.745966692414834
b.Norm(MATRIX_NORM_FROBENIUS)=7.745966692414834
b.Norm(MATRIX_NORM_INF)9.0
b.Norm(MATRIX_NORM_MINUS_INF)2.0
b.Norm(MATRIX_NORM_P1)=)7.0
b.Norm(MATRIX_NORM_MINUS_P1)=6.0
b.Norm(MATRIX_NORM_P2)=7.348469228349533
b.Norm(MATRIX_NORM_MINUS_P2)=1.857033188519056e-16
*/
```

Python example:

```
import numpy as np
from numpy import linalg as LA
a = np.arange(9) - 4
print("a \n",a)
b = a.reshape((3, 3))
print("b \n",b)
print("LA.norm(b)=",LA.norm(b))
print("LA.norm(b, 'fro')=",LA.norm(b, 'fro'))
print("LA.norm(b, np.inf)=",LA.norm(b, np.inf))
print("LA.norm(b, -np.inf)=",LA.norm(b, -np.inf))
print("LA.norm(b, 1)=",LA.norm(b, 1))
print("LA.norm(b, -1)=",LA.norm(b, -1))
print("LA.norm(b, 2)=",LA.norm(b, 2))
print("LA.norm(b, -2)=",LA.norm(b, -2))

a
 [-4 -3 -2 -1  0  1  2  3  4]
b
 [[-4 -3 -2]
 [-1  0  1]
 [ 2  3  4]]
LA.norm(b)= 7.745966692414834
LA.norm(b, 'fro')= 7.745966692414834
LA.norm(b, np.inf)= 9.0
LA.norm(b, -np.inf)= 2.0
LA.norm(b, 1)= 7.0
LA.norm(b, -1)= 6.0
LA.norm(b, 2)= 7.3484692283495345
LA.norm(b, -2)= 1.857033188519056e-16
```

Cond

Compute the condition number of a matrix.

```
double matrix::Cond(
  const ENUM_MATRIX_NORM  norm      // matrix norm
   );
```

Parameters

norm

[in] Order of the norm from ENUM_MATRIX_NORM

Return Value

The condition number of the matrix. May be infinite.

Note

The condition number of x is defined as the norm of x times the norm of the inverse of x [1]. The norm can be the usual L2-norm (root-of-sum-of-squares) or one of a number of other matrix norms.

The condition umber is the K value equal to the product of the matrix A norms and its inverse. Matrices with a high condition number are called ill-conditioned. Those with a low condition number are called well-conditioned. The inverse matrix is obtained using pseudo-inversion, so as not to be limited by the condition of squareness and non-singularity of the matrix.

An exception is the spectral condition number.

A simple algorithm for calculating the spectral condition number in MQL5:

```
double MatrixCondSpectral(matrix& a)
  {
   double norm=0.0;
   vector v=a.Spectrum();

   if(v.Size()>0)
     {
      double max_norm=v[0];
      double min_norm=v[0];
      for(ulong i=1; i<v.Size(); i++)
        {
         double real=MathAbs(v[i]);
         if(max_norm<real)
            max_norm=real;
         if(min_norm>real)
            min_norm=real;
        }
      max_norm=MathSqrt(max_norm);
      min_norm=MathSqrt(min_norm);
      if(min_norm>0.0)
         norm=max_norm/min_norm;
     }

   return(norm);
  }
```

MQL5 example:

```
  matrix a= {{1, 0, -1}, {0, 1, 0}, { 1, 0, 1}};
  Print("a.Cond(MATRIX_NORM_P2)=", a.Cond(MATRIX_NORM_P2));
  Print("a.Cond(MATRIX_NORM_FROBENIUS)=", a.Cond(MATRIX_NORM_FROBENIUS));
  Print("a.Cond(MATRIX_NORM_INF)=", a.Cond(MATRIX_NORM_INF));
  Print("a.Cond(MATRIX_NORM_MINUS_INF)=", a.Cond(MATRIX_NORM_MINUS_INF));
  Print("a.Cond(MATRIX_NORM_P1)=)", a.Cond(MATRIX_NORM_P1));
  Print("a.Cond(MATRIX_NORM_MINUS_P1)=", a.Cond(MATRIX_NORM_MINUS_P1));
  Print("a.Cond(MATRIX_NORM_P2)=", a.Cond(MATRIX_NORM_P2));
  Print("a.Cond(MATRIX_NORM_MINUS_P2)=", a.Cond(MATRIX_NORM_MINUS_P2));

  /*
  matrix a
  [[1,0,-1]
  [0,1,0]
  [1,0,1]]
  a.Cond(MATRIX_NORM_P2)=1.414213562373095
  a.Cond(MATRIX_NORM_FROBENIUS)=3.162277660168379
  a.Cond(MATRIX_NORM_INF)=2.0
  a.Cond(MATRIX_NORM_MINUS_INF)=0.9999999999999997
  a.Cond(MATRIX_NORM_P1)=)2.0
  a.Cond(MATRIX_NORM_MINUS_P1)=0.9999999999999998
  a.Cond(MATRIX_NORM_P2)=1.414213562373095
  a.Cond(MATRIX_NORM_MINUS_P2)=0.7071067811865472
  */
```

Python example:

```
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 0, -1], [0, 1, 0], [1, 0, 1]])
print("a \n",a)
print("LA.cond(a)=",LA.cond(a))
print("LA.cond(a, 'fro')=",LA.cond(a, 'fro'))
print("LA.cond(a, np.inf)=",LA.cond(a, np.inf))
print("LA.cond(a, -np.inf)=",LA.cond(a, -np.inf))
print("LA.cond(a, 1)=",LA.cond(a, 1))
print("LA.cond(a, -1)=",LA.cond(a, -1))
print("LA.cond(a, 2)=",LA.cond(a, 2))
print("LA.cond(a, -2)=",LA.cond(a, -2))

a
 [[ 1  0 -1]
 [ 0  1  0]
 [ 1  0  1]]
LA.cond(a)= 1.4142135623730951
LA.cond(a, 'fro')= 3.1622776601683795
LA.cond(a, np.inf)= 2.0
LA.cond(a, -np.inf)= 1.0
LA.cond(a, 1)= 2.0
LA.cond(a, -1)= 1.0
LA.cond(a, 2)= 1.4142135623730951
LA.cond(a, -2)= 0.7071067811865475
```

Det

Compute the determinant of a square invertible matrix.

```
double matrix::Det()
```

Return Value

Determinant of matrix.

Note

2nd and 3rd order matrix determinants are calculated according to the Sarrus rule. d2=a11*a22-a12*a21; d3=a11*a22*a33+a12*a23*a31+a13*a21*a32-a13*a22*a31-a11*a23*a32-a12*a21*a33

The determinant is calculated by the Gaussian method by reducing the matrix to an upper triangular form. The determinant of an upper triangular matrix is equal to the product of the main diagonal elements.

If at least one matrix row or column is zero, the determinant is zero.

If two or more matrix rows or columns are linearly dependent, its determinant is zero.

The determinant of a matrix is equal to the product of its eigenvalues.

MQL5 example:

```
   matrix m={{1,2},{3,4}};
   double det=m.Det();
   Print("matrix m\n",m);
   Print("det(m)=",det);
   /*
   matrix m
   [[1,2]
    [3,4]]
   det(m)=-2.0
   */
```

Python example:

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
print('a \n',a)
print('nnp.linalg.det(a) \n',np.linalg.det(a))

a
 [[1 2]
 [3 4]]
```

```
np.linalg.det(a)
 -2.0000000000000004
```

SLogDet

Compute the sign and logarithm of a matrix determinant.

```
double matrix::SLogDet(
  int&   sign       // sign
   );
```

Parameters

sign

[out] The sign of the determinant. If the sign is even, the determinant is positive.

Return Value

A number representing the sign of the determinant.

Note

The determinant is calculated by the Gaussian method by reducing the matrix to an upper triangular form. The determinant of an upper triangular matrix is equal to the product of the main diagonal elements. The logarithm of a product is equal to the sum of the logarithms. Therefore, in case of an overflow when calculating the determinant, you can use the SLogDet method.

If the sign is even, the determinant is positive.

Example

```
a = np.array([[1, 2], [3, 4]])
(sign, logdet) = np.linalg.slogdet(a)
(sign, logdet) (-1, 0.69314718055994529) # may vary  sign * np.exp(logdet) -2.0
```

Rank

Return matrix rank using the Gaussian method.

```
int  Rank()
```

Return Value

Rank of matrix.

Note

The rank of a system of rows (or columns) of a matrix A that has m rows and n columns is the maximum number of linearly independent rows (or columns). Several rows (columns) are called linearly independent if none of them can be expressed linearly in terms of others. The rank of the row system is always equal to the rank of the column system. This value is called the rank of the matrix.

MQL5 example:

```
  matrix a=matrix::Eye(4, 4);;
  Print("matrix a \n", a);
  Print("a.Rank()=", a.Rank());

  matrix I=matrix::Eye(4, 4);
  I[3, 3] = 0.;    // rank deficient matrix
  Print("I \n", I);
  Print("I.Rank()=", I.Rank());

  matrix b=matrix::Ones(1, 4);
  Print("b \n", b);
  Print("b.Rank()=", b.Rank());;// 1 dimension - rank 1 unless all 0

  matrix  zeros=matrix::Zeros(4, 1);
  Print("zeros \n", zeros);
  Print("zeros.Rank()=", zeros.Rank());

  /*
  matrix a
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]
  [0,0,0,1]]
  a.Rank()=4

  I
  [[1,0,0,0]
  [0,1,0,0]
  [0,0,1,0]
  [0,0,0,0]]
  I.Rank()=3

  b
  [[1,1,1,1]]
  b.Rank()=1

  zeros
  [[0]
  [0]
  [0]
  [0]]
  zeros.Rank()=0
  */
```

Python example:

```
import numpy as np
from numpy.linalg import matrix_rank
a=(np.eye(4)) # Full rank matrix
print("a \n", a)
print("matrix_rank(a)=",matrix_rank(a))
I=np.eye(4)
I[-1,-1] = 0. # rank deficient matrix
print("I \n",I)
print("matrix_rank(I)=",matrix_rank(I))

b=np.ones((4,))
print("b \n",b)
print("matrix_rank(b)=",matrix_rank(b)) # 1 dimension - rank 1 unless all 0

zeros=np.zeros((4,))
print("zeroes \n",zeros)
print("matrix_rank(zeros)=",matrix_rank(zeros))

a
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
matrix_rank(a)= 4

I
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
```

```
matrix_rank(I)= 3

b
 [1. 1. 1. 1.]
matrix_rank(b)= 1

zeroes
 [0. 0. 0. 0.]
matrix_rank(zeros)= 0
```

Trace

Return the sum along diagonals of the matrix.

```
double matrix::Trace()
```

Return Value

The sum along the diagonal.

Note

The trace of a matrix is equal to the sum of its eigenvalues.

MQL5 example:

```
  matrix a= {{0, 1, 2, 3, 4, 5, 6, 7, 8}};
  a.Reshape(3, 3);
  Print("matrix a \n", a);
  Print("a.Trace() \n", a.Trace());

  /*
  matrix a
  [[0,1,2]
  [3,4,5]
  [6,7,8]]
  a.Trace()
  12.0
  */
```

Python example:

```
a = np.arange(9).reshape((3,3))
print('a \n',a)
print('np.trace(a) \n',np.trace(a))

a
 [[0 1 2]
 [3 4 5]
 [6 7 8]]

np.trace(a)
 12
```

Spectrum

Compute spectrum of a matrix as the set of its eigenvalues from the product AT*A.

```
vector matrix::Spectrum()
```

Return Value

Spectrum of a matrix as a vector of matrix eigenvalues.

Example

```
double MatrixCondSpectral(matrix& a)
  {
   double norm=0.0;
   vector v=a.Spectrum();

   if(v.Size()>0)
     {
      double max_norm=v[0];
      double min_norm=v[0];
      for(ulong i=1; i<v.Size(); i++)
        {
         double real=MathAbs(v[i]);
         if(max_norm<real)
            max_norm=real;
         if(min_norm>real)
            min_norm=real;
        }
      max_norm=MathSqrt(max_norm);
      min_norm=MathSqrt(min_norm);
      if(min_norm>0.0)
         norm=max_norm/min_norm;
     }

   return(norm);
  }
```

Matrix methods for solving systems of linear equations

Methods for solving systems of linear equations and calculating the inverse matrix.

Function	Action
Solve	Solve a linear matrix equation, or system of linear algebraic equations
LstSq	Return the least-squares solution of linear algebraic equations (for non-square or degenerate matrices)
Inv	Compute the (multiplicative) inverse of a square non-degenerate matrix by the Jordan-Gauss method
PInv	Compute the pseudo-inverse of a matrix by the Moore-Penrose method

Solve

Solve a linear matrix equation, or system of linear algebraic equations.

```
vector matrix::Solve(
  const vector  b       // ordinate or 'dependent variable' values
   );
```

Parameters

b

[in] Ordinate or 'dependent variable' values. (Vector of free terms).

Return Value

Vector with solution to the system a * x = b.

Note

If at least one matrix row or column is zero, the system has no solution.

If two or more matrix rows or columns are linearly dependent, the system has no solution.

Example

```
//--- SLAE solution
   vector_x=matrix_a.Solve(vector_b);
//--- check if a * x = b
   result_vector=matrix_a.MatMul(vector_x);
   errors=vector_b.Compare(result_vector,1e-12);
```

LstSq

Return the least-squares solution of linear algebraic equations (for non-square or degenerate matrices).

```
vector matrix::LstSq(
  const vector  b      // ordinate or 'dependent variable' values
   );
```

Parameters

b

[in] Ordinate or 'dependent variable' values. (Vector of free terms)

Return Value

Vector with solution to the system a * x = b. This is true only for systems that have an exact solution.

Example

```
   matrix a={{3, 2},
             {4,-5},
             {3, 3}};
   vector b={7,40,3};
//---
   vector x=a.LstSq(b);
//--- check, must be [5, -4]
   Print("x=", x);
//--- check, must be [7, 40, 3]
   vector b1=a.MatMul(x);
   Print("b1=",b1);

/*
  x=[5.000000000000002,-4]
  b1=[7.000000000000005,40.00000000000001,3.000000000000005]
*/
```

Inv

Compute the multiplicative inverse of a square invertible matrix by the Jordan-Gauss method.

```
matrix matrix::Inv()
```

Return Value

Multiplicative inverse of the matrix.

Note

The product of the original matrix and the inverse matrix is the identity matrix.

If at least one matrix row or column is zero, the inverse matrix cannot be obtained.

If two or more matrix rows or columns are linearly dependent, the inverse matrix cannot be obtained.

Example

```
int TestInverse(const int size_m)
  {
   int     i,j,errors=0;
   matrix matrix_a(size_m,size_m);
//--- populate the square matrix
   MatrixTestFirst(matrix_a);
//--- microseconds will be measured
   ulong t1=GetMicrosecondCount();
//--- get the inverse matrix
   matrix inverse=matrix_a.Inv();
//--- measure
   ulong t2=GetMicrosecondCount();
//--- check the correctness
   matrix identity=matrix_a.MatMul(inverse);
//---
   for(i=0; i<size_m; i++)
     {
      for(j=0; j<size_m; j++)
        {
         double value;
         //--- ones must be along the diagonal
         if(i==j)
            value=1.0;
         else
            value=0.0;
         if(MathClassify(identity[i][j])>FP_ZERO)
            errors++;
         else
```

```
           {
            if(identity[i][j]!=value)
              {
               double diff=MathAbs(identity[i][j]-value);
               //--- too many multiplications and devisions, so reduce the check accu
               if(diff>1e-9)
                  errors++;
              }
           }
        }
     }
//---
   double elapsed_time=double(t2-t1)/1000.0;
   printf("Inversion of matrix %d x %d  %s  errors=%d  time=%.3f ms",size_m,size_m,er
   return(errors);
  }
```

PInv

Compute the pseudo-inverse of a matrix by the Moore-Penrose method.

```
matrix matrix::PInv()
```

Return Value

The pseudo-inverse of matrix.

Example

```
int TestPseudoInverse(const int size_m, const int size_k)
  {
   matrix matrix_a(size_m,size_k);
   matrix matrix_inverted(size_k,size_m);
   matrix matrix_temp;
   matrix matrix_a2;
//--- fill the matrix
   MatrixTestFirst(matrix_a);
//--- invert
   matrix_inverted=matrix_a.PInv();
//--- check the correctness
   int errors=0;
//--- A * A+ * A = A   (A+ is a pseudo-inverse of A)
   matrix_temp=matrix_a.MatMul(matrix_inverted);
   matrix_a2=matrix_temp.MatMul(matrix_a);
   errors=(int)matrix_a.CompareByDigits(matrix_a2,10);

   printf("PseudoInversion %s matrix_size  %d x %d  errors=%d",errors==0?"passed":"fa
//---
   return(errors);
  }
```

Machine learning

These methods are used in machine learning.

The neural network activation function determines the output value of a neuron depending on the weighted sum of inputs. The selection of the activation function has a big impact on the neural network performance. Different model parts (layers) can use different activation functions.

In addition to all known activation functions, MQL5 also offers their derivatives. Function derivatives enable an efficient update of model parameters based on the error received in learning.

A neural network aims at finding an algorithm that minimizes the error in learning, for which the loss function is used. The value of the loss function indicates by how much the value predicted by the model deviates from the real one. Different loss functions are used depending on the problem. For example, Mean Squared Error (MSE) is used for regression problems, and Binary Cross-Entropy (BCE) is used for binary classification purposes.

Function	Action
Activation	Compute activation function values and write them to the passed vector/matrix
Derivative	Compute activation function derivative values and write them to the passed vector/matrix
Loss	Compute loss function values and write them to the passed vector/matrix
LossGradient	Compute a vector or matrix of loss function gradients
RegressionMetric	Compute the regression metric as the deviation error from the regression line constructed on the specified data array
ConfusionMatrix	Compute confusion matrix. The method is applied to the vector of predicted values
ConfusionMatrixMultilabel	Compute confusion matrix for each label. The method is applied to the vector of predicted values
ClassificationMetric	Compute the classification metric to evaluate the quality of the predicted data compared to the true data The method is applied to the vector of predicted values
ClassificationScore	Compute the classification metric to evaluate the quality of the predicted data compared to the true data
PrecisionRecall	Compute values to construct a precision-recall curve. Similarly to ClassificationScore, this method is applied to the vector of true values
ReceiverOperatingCharacteristic	Compute values to construct the Receiver Operating Characteristic (ROC) curve. Similarly to ClassificationScore, this method is applied to the vector of true values.

Example

This example demonstrates the training of a model using matrix operations. The model is trained for the function (a + b + c)^2 / (a^2 + b^2 + c^2). We input the initial data matrix, in which a, b and c are contained in different columns. The function result is obtained at the model output.

```
matrix weights1, weights2, weights3;               // matrices of weights
matrix output1, output2, result;                   // matrices of neural layer outputs
input int layer1 = 200;                            // the size of the first hidden lay
input int layer2 = 200;                            // the size of the second hidden la
input int Epochs = 20000;                          // the number of training epochs
input double lr = 3e-6;                            // learning rate
input ENUM_ACTIVATION_FUNCTION ac_func = AF_SWISH; // activation function
//+------------------------------------------------------------------+
//| Script start function                                            |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int train = 1000;    // training sample size
   int test = 10;       // test sample size
   matrix m_data, m_target;
//--- generate a training sample
   if(!CreateData(m_data, m_target, train))
      return;
//--- train the model
   if(!Train(m_data, m_target, Epochs))
      return;
//--- generate a test sample
   if(!CreateData(m_data, m_target, test))
      return;
//--- test the model
   Test(m_data, m_target);
  }
//+------------------------------------------------------------------+
//| Sample generation method                                         |
//+------------------------------------------------------------------+
bool CreateData(matrix &data, matrix &target, const int count)
  {
//--- initialize the initial data and result matrices
   if(!data.Init(count, 3) || !target.Init(count, 1))
      return false;
//--- fill the initial data matrix with random values
   data.Random(-10, 10);
//--- calculate the target values for the training sample
   vector X1 = MathPow(data.Col(0) + data.Col(1) + data.Col(1), 2);
   vector X2 = MathPow(data.Col(0), 2) + MathPow(data.Col(1), 2) + MathPow(data.Col(2)
   if(!target.Col(X1 / X2, 0))
      return false;
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
//| Model training method                                            |
//+------------------------------------------------------------------+
bool Train(matrix &data, matrix &target, const int epochs = 10000)
  {
//--- create the model
   if(!CreateNet())
      return false;
//--- train the model
   for(int ep = 0; ep < epochs; ep++)
     {
      //--- feedforward pass
      if(!FeedForward(data))
         return false;
      PrintFormat("Epoch %d, loss %.5f", ep, result.Loss(target, LOSS_MSE));
      //--- backpropagation and update of weight matrix
      if(!Backprop(data, target))
         return false;
     }
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
//| Model creation method                                            |
//+------------------------------------------------------------------+
bool CreateNet()
  {
//--- initialize weight matrices
   if(!weights1.Init(4, layer1) || !weights2.Init(layer1 + 1, layer2) || !weights3.In
      return false;
//--- fill the weight matrices with random values
   weights1.Random(-0.1, 0.1);
   weights2.Random(-0.1, 0.1);
   weights3.Random(-0.1, 0.1);
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
//| Feedforward method                                               |
//+------------------------------------------------------------------+
bool FeedForward(matrix &data)
  {
//--- check the initial data size
   if(data.Cols() != weights1.Rows() - 1)
      return false;
//--- calculate the first neural layer
   matrix temp = data;
   if(!temp.Resize(temp.Rows(), weights1.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights1.Rows() - 1))
      return false;
   output1 = temp.MatMul(weights1);
//--- calculate the activation function
   if(!output1.Activation(temp, ac_func))
      return false;
//--- calculate the second neural layer
   if(!temp.Resize(temp.Rows(), weights2.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights2.Rows() - 1))
      return false;
   output2 = temp.MatMul(weights2);
//--- calculate the activation function
   if(!output2.Activation(temp, ac_func))
      return false;
//--- calculate the third neural layer
   if(!temp.Resize(temp.Rows(), weights3.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights3.Rows() - 1))
      return false;
   result = temp.MatMul(weights3);
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
//| Backpropagation method                                           |
//+------------------------------------------------------------------+
bool Backprop(matrix &data, matrix &target)
  {
//--- check the size of the matrix of target values
   if(target.Rows() != result.Rows() ||
      target.Cols() != result.Cols())
      return false;
//--- determine the deviation of calculated values from target
   matrix loss = (target - result) * 2;
//--- propagate the gradient to the previous layer
   matrix gradient = loss.MatMul(weights3.Transpose());
//--- update the wight matrix of the last layer
   matrix temp;
   if(!output2.Activation(temp, ac_func))
      return false;
   if(!temp.Resize(temp.Rows(), weights3.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights3.Rows() - 1))
      return false;
   weights3 = weights3 + temp.Transpose().MatMul(loss) * lr;
//--- adjust the error gradient by the derivative of the activation function
   if(!output2.Derivative(temp, ac_func))
      return false;
   if(!gradient.Resize(gradient.Rows(), gradient.Cols() - 1))
      return false;
   loss = gradient * temp;
```

```
//--- propagate the gradient to a lower layer
   gradient = loss.MatMul(weights2.Transpose());
//--- update the weight matrix of the second hidden layer
   if(!output1.Activation(temp, ac_func))
      return false;
   if(!temp.Resize(temp.Rows(), weights2.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights2.Rows() - 1))
      return false;
   weights2 = weights2 + temp.Transpose().MatMul(loss) * lr;
//--- adjust the error gradient by the derivative of the activation function
   if(!output1.Derivative(temp, ac_func))
      return false;
   if(!gradient.Resize(gradient.Rows(), gradient.Cols() - 1))
      return false;
   loss = gradient * temp;
//--- update the weight matrix of the first hidden layer
   temp = data;
   if(!temp.Resize(temp.Rows(), weights1.Rows()) ||
      !temp.Col(vector::Ones(temp.Rows()), weights1.Rows() - 1))
      return false;
   weights1 = weights1 + temp.Transpose().MatMul(loss) * lr;
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
//| Model testing method                                             |
//+------------------------------------------------------------------+
bool Test(matrix &data, matrix &target)
  {
//--- feedfarward on test data
   if(!FeedForward(data))
      return false;
//--- log the model calculation results and true values
   PrintFormat("Test loss %.5f", result.Loss(target, LOSS_MSE));
   ulong total = data.Rows();
   for(ulong i = 0; i < total; i++)
      PrintFormat("(%.2f + %.2f + %.2f)^2 / (%.2f^2 + %.2f^2 + %.2f^2) =  Net %.2f, Ta
                  data[i, 0], data[i, 1], data[i, 2], result[i, 0], target[i, 0]);
//--- return result
   return true;
  }
//+------------------------------------------------------------------+
```

Activation

Compute activation function values and write them to the passed vector/matrix.

```
bool vector::Activation(
  vector&                   vect_out,      // vector to get values
  ENUM_ACTIVATION_FUNCTION  activation,    // activation function
   ...                                     // additional parameters
   );


bool matrix::Activation(
  matrix&                   matrix_out,    // matrix to get values
  ENUM_ACTIVATION_FUNCTION  activation     // activation function
   );


bool matrix::Activation(
  matrix&                   matrix_out,    // matrix to get values
  ENUM_ACTIVATION_FUNCTION  activation,    // activation function
  ENUM_MATRIX_AXIS          axis,          // axis
   ...                                     // additional parameters
   );
```

Parameters

vect_out/matrix_out

[out] Vector or matrix to get the computed values of the activation function.

activation

[in] Activation function from the ENUM_ACTIVATION_FUNCTION enumeration.

axis

[in] ENUM_MATRIX_AXIS enumeration value (AXIS_HORZ — horizontal axis, AXIS_VERT — vertical axis).

...

[in] Additional parameters required for some activation functions. If no parameters are specified, default values are used.

Return Value

Returns true if successful, otherwise - false.

Additional Parameters

Some activation functions accept additional parameters. If no parameters are specified, default values are used

```
AF_ELU  (Exponential Linear Unit)
  double alpha=1.0

Activation function: if(x>=0) f(x) = x
                   else f(x) = alpha * (exp(x)-1)


AF_LINEAR
  double alpha=1.0
  double beta=0.0

Activation function: f(x) = alpha*x + beta


AF_LRELU   (Leaky REctified Linear Unit)
  double alpha=0.3

Activation function: if(x>=0) f(x)=x
                   else f(x) = alpha*x


AF_RELU  (REctified Linear Unit)
  double alpha=0.0
  double max_value=0.0
  double treshold=0.0

Activation function: if(alpha==0) f(x) = max(x,0)
                   else if(x>max_value) f(x) = x
                   else f(x) = alpha*(x - treshold)


AF_SWISH
  double beta=1.0

Activation function: f(x) = x / (1+exp(-x*beta))


AF_TRELU   (Thresholded REctified Linear Unit)
  double theta=1.0

Activation function: if(x>theta) f(x) = x
                   else f(x) = 0


AF_PRELU   (Parametric REctified Linear Unit)
  double alpha[] - learned array of coeefficients
```

```
Activation function: if(x[i]>=0) f(x)[i] = x[i]
                     else f(x)[i] = alpha[i] * x[i]
```

Note

In artificial neural networks, the activation function of a neuron determines the output signal, which is defined by an input signal or a set of input signals. The selection of the activation function has a big impact on the neural network performance. Different model parts (layers) can use different activation functions.

Examples of using additional parameters:

```
vector x={0.1, 0.4, 0.9, 2.0, -5.0, 0.0, -0.1};
vector y;

x.Activation(y,AF_ELU);
Print(y);
x.Activation(y,AF_ELU,2.0);
Print(y);

Print("");
x.Activation(y,AF_LINEAR);
Print(y);
x.Activation(y,AF_LINEAR,2.0);
Print(y);
x.Activation(y,AF_LINEAR,2.0,5.0);
Print(y);

Print("");
x.Activation(y,AF_LRELU);
Print(y);
x.Activation(y,AF_LRELU,1.0);
Print(y);
x.Activation(y,AF_LRELU,0.1);
Print(y);

Print("");
x.Activation(y,AF_RELU);
Print(y);
x.Activation(y,AF_RELU,2.0,0.5);
Print(y);
x.Activation(y,AF_RELU,2.0,0.5,1.0);
Print(y);

Print("");
x.Activation(y,AF_SWISH);
Print(y);
```

```
   x.Activation(y,AF_SWISH,2.0);
   Print(y);

   Print("");
   x.Activation(y,AF_TRELU);
   Print(y);
   x.Activation(y,AF_TRELU,0.3);
   Print(y);

   Print("");
   vector a=vector::Full(x.Size(),2.0);
   x.Activation(y,AF_PRELU,a);
   Print(y);

/*  Results
   [0.1,0.4,0.9,2,-0.993262053000915,0,-0.095162581964040]
   [0.1,0.4,0.9,2,-1.986524106001829,0,-0.190325163928081]

   [0.1,0.4,0.9,2,-5,0,-0.1]
   [0.2,0.8,1.8,4,-10,0,-0.2]
   [5.2,5.8,6.8,9,-5,5,4.8]

   [0.1,0.4,0.9,2,-1.5,0,-0.03]
   [0.1,0.4,0.9,2,-5,0,-0.1]
   [0.1,0.4,0.9,2,-0.5,0,-0.01]

   [0.1,0.4,0.9,2,0,0,0]
   [0.2,0.8,0.9,2,-10,0,-0.2]
   [-1.8,-1.2,0.9,2,-12,-2,-2.2]

   [0.052497918747894,0.239475064044981,0.6398545523625035,1.761594155955765,-0.03346
   [0.054983399731247,0.275989792451045,0.7723340415895611,1.964027580075817,-0.00022

   [0,0,0,2,0,0,0]
   [0,0.4,0.9,2,0,0,0]

   [0.1,0.4,0.9,2,-10,0,-0.2]
*/
```

Derivative

Compute activation function derivative values and write them to the passed vector/matrix

```
bool vector::Derivative(
  vector&                    vect_out,      // vector to get values
  ENUM_ACTIVATION_FUNCTION   activation,    // activation function
   ...                                      // additional parameters
   );


bool matrix::Derivative(
  matrix&                    matrix_out,    // matrix to get values
  ENUM_ACTIVATION_FUNCTION   activation,    // activation function
   );


bool matrix::Derivative(
  matrix&                    matrix_out,    // matrix to get values
  ENUM_ACTIVATION_FUNCTION   activation,    // activation function
  ENUM_MATRIX_AXIS           axis,          // axis
   ...                                      // additional parameters
   );
```

Parameters

vect_out/matrix_out

[out] Vector or matrix to get the computed values of the derivative of the activation function.

activation

[in] Activation function from the ENUM_ACTIVATION_FUNCTION enumeration.

axis

[in] ENUM_MATRIX_AXIS enumeration value (AXIS_HORZ — horizontal axis, AXIS_VERT — vertical axis).

...

[in] Additional parameters are the same as that of the activation functions. Only some activation functions accept additional parameters. If no parameters are specified, default values are used.

Return Value

Returns true if successful, otherwise - false.

Note

Function derivatives enable an efficient update of model parameters based on the error received in learning during the error backpropagation.

Loss

Compute the value of the loss function.

```
double vector::Loss(
  const vector&        vect_true,     // vector of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function
   ...                                // additional parameter
   );


double matrix::Loss(
  const matrix&        matrix_true,   // matrix of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function
   );


double matrix::Loss(
  const matrix&        matrix_true,   // matrix of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function
  ENUM_MATRIX_AXIS     axis,          // axis
   ...                                // additional parameter
   );
```

Parameters

vect_true/matrix_true

[in] Vector or matrix of true values.

loss

[in] Loss function from the ENUM_LOSS_FUNCTION enumeration.

axis

[in] ENUM_MATRIX_AXIS enumeration value (AXIS_HORZ — horizontal axis, AXIS_VERT — vertical axis).

...

[in] Additional parameter 'delta' can only be used by the Hubert loss function (LOSS_HUBER)

Return Value

double value.

How the 'delta' parameter is used in the Hubert loss function (LOSS_HUBER)

```
   double delta = 1.0;
   double error = fabs(y - x);
   if(error<delta)
      loss = 0.5 * error^2;
   else
```

```
      loss = 0.5 * delta^2 + delta * (error - delta);
```

Note

A neural network aims at finding the algorithms that minimize the error on the training sample, for which the loss function is used.

The value of the loss function indicates by how much the value predicted by the model deviates from the real one.

Different loss functions are used depending on the problem. For example, Mean Squared Error (MSE) is used for regression problems, and Binary Cross-Entropy (BCE) is used for binary classification purposes.

Example of calling the Hubert loss function:

```
   vector y_true = {0.0, 1.0, 0.0, 0.0};
   vector y_pred = {0.6, 0.4, 0.4, 0.6};
   double loss=y_pred.Loss(y_true,LOSS_HUBER);
   Print(loss);
   double loss2=y_pred.Loss(y_true,LOSS_HUBER,0.5);
   Print(loss2);

/* Result
   0.155
   0.15125
*/
```

LossGradient

Compute a vector or matrix of loss function gradients.

```
vector vector::LossGradient(
  const vector&        vect_true,     // vector of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function type
   ...                                // additional parameter
   );


matrix matrix::LossGradient(
  const matrix&        matrix_true,   // matrix of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function
   );


matrix matrix::LossGradient(
  const matrix&        matrix_true,   // matrix of true values
  ENUM_LOSS_FUNCTION   loss,          // loss function
  ENUM_MATRIX_AXIS     axis,          // axis
   ...                                // additional parameter
   );
```

Parameters

vect_true/matrix_true

[in] Vector or matrix of true values.

loss

[in] Loss function from the ENUM_LOSS_FUNCTION enumeration.

axis

[in] ENUM_MATRIX_AXIS enumeration value (AXIS_HORZ — horizontal axis, AXIS_VERT — vertical axis).

...

[in] Additional parameter 'delta' can only be used by the Hubert loss function (LOSS_HUBER)

Return Value

Vector or matrix of loss function gradient values. The gradient is the partial derivative with respect to dx (x is the predicted value) of the loss function at a given point.

Note

Gradients are used in neural networks to adjust the weight matrix weights during backpropagation, when training the model.

A neural network aims at finding the algorithms that minimize the error on the training sample, for which the loss function is used.

Different loss functions are used depending on the problem. For example, Mean Squared Error (MSE) is used for regression problems, and Binary Cross-Entropy (BCE) is used for binary classification purposes.

Example of calculating loss function gradients

```
   matrixf y_true={{ 1, 2, 3, 4 },
                   { 5, 6, 7, 8 },
                   { 9,10,11,12 }};
   matrixf y_pred={{ 1, 2, 3, 4 },
                   {11,10, 9, 8 },
                   { 5, 6, 7,12 }};
   matrixf loss_gradient =y_pred.LossGradient(y_true,LOSS_MAE);
   matrixf loss_gradienth=y_pred.LossGradient(y_true,LOSS_MAE,AXIS_HORZ);
   matrixf loss_gradientv=y_pred.LossGradient(y_true,LOSS_MAE,AXIS_VERT);
   Print("loss gradients\n",loss_gradient);
   Print("loss gradients on horizontal axis\n",loss_gradienth);
   Print("loss gradients on vertical axis\n",loss_gradientv);

/* Result
   loss gradients
   [[0,0,0,0]
    [0.083333336,0.083333336,0.083333336,0]
    [-0.083333336,-0.083333336,-0.083333336,0]]
   loss gradients on horizontal axis
   [[0,0,0,0]
    [0.33333334,0.33333334,0.33333334,0]
    [-0.33333334,-0.33333334,-0.33333334,0]]
   loss gradients on vertical axis
   [[0,0,0,0]
    [0.25,0.25,0.25,0]
    [-0.25,-0.25,-0.25,0]]
*/
```

RegressionMetric

Compute the regression metric to evaluate the quality of the predicted data compared to the true data

```
double vector::RegressionMetric(
   const vector&           vector_true,   // vector of true values
   ENUM_REGRESSION_METRIC  metric         // metric type
   );

double matrix::RegressionMetric(
   const matrix&           matrix_true,   // matrix of true values
   ENUM_REGRESSION_METRIC  metric         // metric type
);

vector matrix::RegressionMetric(
   const matrix&           matrix_true,   // matrix of true values
   ENUM_REGRESSION_METRIC  metric,        // metric type
   int                     axis           // axis
   );
```

Parameters

vector_true/matrix_true

[in] Vector or matrix of true values.

metric

[in] Metric type from the ENUM_REGRESSION_METRIC enumeration.

axis

[in] Axis. 0 — horizontal axis, 1 — vertical axis.

Return Value

The calculated metric which evaluates the quality of the predicted data compared to the true data.

Note

- REGRESSION_MAE — mean absolute error which represents the absolute differences between predicted values and corresponding true values
- REGRESSION_MSE — mean square error which represents the squared differences between predicted values and corresponding true values
- REGRESSION_RMSE — square root of MSE
- REGRESSION_R2 - 1 — MSE(regression) / MSE(mean)
- REGRESSION_MAPE — MAE as a percentage
- REGRESSION_MSPE — MSE as a percentage
- REGRESSION_RMSLE — RMSE computed on a logarithmic scale

Example:

```
    vector y_true = {3, -0.5, 2, 7};
    vector y_pred = {2.5, 0.0, 2, 8};
//---
    double mse=y_pred.RegressionMetric(y_true,REGRESSION_MSE);
    Print("mse=",mse);
//---
    double mae=y_pred.RegressionMetric(y_true,REGRESSION_MAE);
    Print("mae=",mae);
//---
    double r2=y_pred.RegressionMetric(y_true,REGRESSION_R2);
    Print("r2=",r2);

  /* Result
    mae=0.375
    mse=0.5
    r2=0.9486081370449679
    */
```

ConfusionMatrix

Compute confusion matrix. The method is applied to the vector of predicted values.

```
matrix vector::ConfusionMatrix(
   const vector&        vect_true      // vector of true values
   );


matrix vector::ConfusionMatrix(
   const vector&        vect_true,     // vector of true values
   uint                 label          // label value
   );
```

Parameters

vect_true

[in] Vector of true values.

label

[in] Label value for calculating the confusion matrix.

Return Value

Confusion matrix. If a label value is not specified, a multi-class confusion matrix is returned, where each label is matched to each other label individually. If a label value is specified, a 2 x 2 matrix is returned, in which the specified label is considered positive, while all other labels are negative (ovr, one vs rest).

Note

Confusion matrix C is such that Cij is equal to the number of observations known to be in group i and predicted to be in group j. Thus in binary classification, the count of true negatives (TN) is C00, false negatives (FN) is C10, true positives (TP) is C11 and false positives (FP) is C01.

In other words, the matrix can be graphically represented as follows:

TN	FP
FN	TP

The sizes of the vector of true values and the vector of predicted values should be the same.

Example:

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};
   matrix confusion=y_pred.ConfusionMatrix(y_true);
```

```
    Print(confusion);
    confusion=y_pred.ConfusionMatrix(y_true,0);
    Print(confusion);
    confusion=y_pred.ConfusionMatrix(y_true,1);
    Print(confusion);
    confusion=y_pred.ConfusionMatrix(y_true,2);
    Print(confusion);


/*
  [[3,0,0,0,0,0,0,0,0,0]
   [0,3,0,0,0,0,0,0,0,0]
   [0,0,1,0,1,0,0,1,0,0]
   [0,0,0,1,0,0,0,1,0,0]
   [0,0,1,0,3,0,0,0,0,1]
   [0,0,0,0,0,2,0,0,0,0]
   [1,0,0,0,0,1,1,0,0,0]
   [0,0,0,0,0,0,0,2,0,1]
   [0,0,1,0,0,0,0,0,0,1]
   [0,0,0,0,0,0,0,0,0,4]]
  [[26,1]
   [0,3]]
  [[27,0]
   [0,3]]
  [[25,2]
   [2,1]]
*/
```

ConfusionMatrixMultilabel

Compute confusion matrix for each label. The method is applied to the vector of predicted values.

```
uint vector::ConfusionMatrixMultiLabel(
   const vector&        vect_true,      // vector of true values
   matrix&              confusions[]    // array of calculated confusion matrices
   );
```

Parameters

vect_true

[in] Vector of true values.

confusions

[out] An array of 2 x 2 matrices with computed confusion matrices for each label.

Return Value

Size of the array of calculated confusion matrices. In case of failure, it returns 0

Note

The result array can be dynamic or static. If the array is static, then it must be no less in size than the number of classes.

The sizes of the vector of true values and the vector of predicted values should be the same.

Example:

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};
   matrix label_confusions[12];

   uint   res=y_pred.ConfusionMatrixMultiLabel(y_true,label_confusions);
   Print("res=",res,"  size=",label_confusions.Size());
   for(uint i=0; i<res; i++)
      Print(label_confusions[i]);


/*
  res=10  size=12
  [[26,1]
   [0,3]]
  [[27,0]
   [0,3]]
  [[25,2]
   [2,1]]
   [[28,0]
    [1,1]]
   [[24,1]
    [2,3]]
   [[27,1]
    [0,2]]
   [[27,0]
    [2,1]]
   [[25,2]
    [1,2]]
   [[28,0]
    [2,0]]
   [[23,3]
    [0,4]]
*/
```

ClassificationMetric

Compute the classification metric to evaluate the quality of the predicted data compared to the true data. The method is applied to the vector of predicted values.

```
vector vector::ClassificationMetric(
   const vector&                 vect_true,      // vector of true values
   ENUM_CLASSIFICATION_METRIC metric          // metric type
   );


vector vector::ClassificationMetric(
   const vector&                 vect_true,      // vector of true values
   ENUM_CLASSIFICATION_METRIC metric          // metric type
   ENUM_AVERAGE_MODE             mode            // averaging mode
   );
```

Parameters

vect_true

[in] Vector of true values.

metric

[in] Metric type from the ENUM_CLASSIFICATION_METRIC enumeration. Values other than CLASSIFICATION_TOP_K_ACCURACY, CLASSIFICATION_AVERAGE_PRECISION and CLASSIFICATION_ROC_AUC (used in the ClassificationScore method) are applied.

mode

[in] Averaging mode from the ENUM_AVERAGE_MODE enumeration. Used for the CLASSIFICATION_F1, CLASSIFICATION_JACCARD, CLASSIFICATION_PRECISION and CLASSIFICATION_RECALL metrics.

Return Value

A vector containing the calculated metric. In the case of the AVERAGE_NONE averaging mode, the vector contains metric values for each class without averaging. (For example, in case of the binary classification, this would be two metrics for 'false' and 'true' respectively).

Note about averaging modes

AVERAGE_BINARY is only meaningful for binary classification.

AVERAGE_MICRO – calculate metrics globally by counting the total true positives, false negatives and false positives.

AVERAGE_MACRO – calculate metrics for each label and find their unweighted mean. This does not take label imbalance into account.

AVERAGE_WEIGHTED – calculate metrics for each label and find their average weighted by support (the number of true instances for each label). This alters ‘macro’ to account for label imbalance; it can result in an F-score that is not between precision and recall.

Note

In case of binary classification, we can input not only an n x 2 matrix, where the first column contains probabilities for a negative label, and the second column contains probabilities for a positive label, but also a matrix consisting of one column with positive probabilities. This is because binary classification models can return either two probabilities or one probability for a positive label.

Example:

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};

   vector accuracy=y_pred.ClassificationMetric(y_true,CLASSIFICATION_ACCURACY);
   Print("accuracy=",accuracy);
   vector balanced=y_pred.ClassificationMetric(y_true,CLASSIFICATION_BALANCED_ACCURAC
   Print("balanced=",balanced);
   Print("");

   vector f1_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_MICRO
   Print("f1_micro=",f1_micro);
   vector f1_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_MACRO
   Print("f1_macro=",f1_macro);
   vector f1_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_WE
   Print("f1_weighted=",f1_weighted);
   vector f1_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_F1,AVERAGE_NONE);
   Print("f1_none=",f1_none);
   Print("");

   vector jaccard_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVE
   Print("jaccard_micro=",jaccard_micro);
   vector jaccard_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVE
   Print("jaccard_macro=",jaccard_macro);
   vector jaccard_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,
   Print("jaccard_weighted=",jaccard_weighted);
   vector jaccard_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_JACCARD,AVER
   Print("jaccard_none=",jaccard_none);
   Print("");

   vector precision_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION
   Print("precision_micro=",precision_micro);
   vector precision_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION
   Print("precision_macro=",precision_macro);
   vector precision_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECIS
   Print("precision_weighted=",precision_weighted);
   vector precision_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_PRECISION,
   Print("precision_none=",precision_none);
   Print("");
   vector recall_micro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERA
   Print("recall_micro=",recall_micro);
   vector recall_macro=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERA
   Print("recall_macro=",recall_macro);
   vector recall_weighted=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AV
   Print("recall_weighted=",recall_weighted);
   vector recall_none=y_pred.ClassificationMetric(y_true,CLASSIFICATION_RECALL,AVERAG
   Print("recall_none=",recall_none);
   Print("");

//--- binary classification
   vector y_pred_bin={0,1,0,1,1,0,0,0,1};
   vector y_true_bin={1,0,0,0,1,0,1,1,1};

   vector f1_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_F1,AVERAGE
   Print("f1_bin=",f1_bin);
   vector jaccard_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_JACCA
   Print("jaccard_bin=",jaccard_bin);
   vector precision_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_PRE
   Print("precision_bin=",precision_bin);
   vector recall_bin=y_pred_bin.ClassificationMetric(y_true_bin,CLASSIFICATION_RECALL,
   Print("recall_bin=",recall_bin);


/*
  accuracy=[0.6666666666666666]
  balanced=[0.6433333333333333]

  f1_micro=[0.6666666666666666]
  f1_macro=[0.6122510822510823]
  f1_weighted=[0.632049062049062]
  f1_none=[0.8571428571428571,1,0.3333333333333333,0.6666666666666666,0.66666666666666

  jaccard_micro=[0.5]
  jaccard_macro=[0.4921428571428572]
  jaccard_weighted=[0.5056349206349205]
  jaccard_none=[0.75,1,0.2,0.5,0.5,0.6666666666666666,0.3333333333333333,0.4,0,0.5714

  precision_micro=[0.6666666666666666]
  precision_macro=[0.6571428571428571]
  precision_weighted=[0.6706349206349207]
  precision_none=[0.75,1,0.3333333333333333,1,0.75,0.6666666666666666,1,0.5,0,0.57142

  recall_micro=[0.6666666666666666]
  recall_macro=[0.6433333333333333]
  recall_weighted=[0.6666666666666666]
  recall_none=[1,1,0.3333333333333333,0.5,0.6,1,0.3333333333333333,0.6666666666666666,

  f1_bin=[0.4444444444444445]
  jaccard_bin=[0.2857142857142857]
  precision_bin=[0.5]
  recall_bin=[0.4]
*/
```

ClassificationScore

Compute the classification metric to evaluate the quality of the predicted data compared to the true data.

Unlike other methods in the Machine Learning section, this one applies to the vector of true values rather than the vector of predicted values.

```
vector vector::ClassificationScore(
   const matrix&                pred_scores,    // matrix containing probability distribu
   ENUM_CLASSIFICATION_METRIC metric          // metric type
   ENUM_AVERAGE_MODE            mode            // averaging mode
   );


vector vector::ClassificationScore(
   const matrix&                pred_scores,    // matrix containing probability distribu
   ENUM_CLASSIFICATION_METRIC metric          // metric type
   int                          param           // additional parameter
   );
```

Parameters

pred_scores

[in] A matrix containing a set of horizontal vectors with probabilities for each class. The number of matrix rows should correspond to the size of the vector of true values.

metric

[in] Metric type from the ENUM_CLASSIFICATION_METRIC enumeration. The CLASSIFICATION_TOP_K_ACCURACY, CLASSIFICATION_AVERAGE_PRECISION and CLASSIFICATION_ROC_AUC values are used.

mode

[in] Averaging mode from the ENUM_AVERAGE_MODE enumeration. Used for the CLASSIFICATION_AVERAGE_PRECISION and CLASSIFICATION_ROC_AUC metrics.

param

[in] In case of the CLASSIFICATION_TOP_K_ACCURACY metric, the integer K value should be specified instead of the averaging mode.

Return Value

A vector containing the calculated metric. In the case of the AVERAGE_NONE averaging mode, the vector contains metric values for each class without averaging. (For example, in case of the binary classification, this would be two metrics for 'false' and 'true' respectively).

Note about averaging modes

AVERAGE_BINARY is only meaningful for binary classification.

AVERAGE_MICRO – calculate metrics globally by considering each element of the label indicator matrix as a label. The label indicator matrix refers to a matrix with a set of probabilities for each label.

AVERAGE_MACRO – calculate metrics for each label and find their unweighted mean. This does not take label imbalance into account.

AVERAGE_WEIGHTED – calculate metrics for each label and find their average weighted by support (the number of true instances for each label).

Note

In case of binary classification, we can input not only an n x 2 matrix, where the first column contains probabilities for a negative label, and the second column contains probabilities for a positive label, but also a matrix consisting of one column with positive probabilities. This is because binary classification models can return either two probabilities or one probability for a positive label.

Example:

```
   vector y_true={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,8,4,2,7,6,8,4,2,3,6};
   //vector y_pred={7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4,2,9,4,9,5,9,2,7,7,0};

//--- label scores          0         1         2         3         4         5
   matrix y_scores={{0.000109, 0.000186, 0.000449, 0.000052, 0.000002, 0.000022, 0.00
                    {0.000091, 0.081956, 0.916816, 0.001106, 0.000006, 0.000002, 0.00
                    {0.000108, 0.972863, 0.003600, 0.000021, 0.010479, 0.000015, 0.00
                    {0.925425, 0.000080, 0.002913, 0.000057, 0.000274, 0.000638, 0.06
                    {0.000060, 0.000126, 0.000006, 0.000000, 0.993513, 0.000000, 0.00
                    {0.000016, 0.982124, 0.000045, 0.000002, 0.008445, 0.000001, 0.00
                    {0.000000, 0.000040, 0.000001, 0.000000, 0.989395, 0.000167, 0.00
                    {0.000795, 0.002938, 0.023447, 0.007418, 0.021838, 0.002476, 0.00
                    {0.000091, 0.000226, 0.000038, 0.000007, 0.000048, 0.854910, 0.06
                    {0.000000, 0.000000, 0.000000, 0.000000, 0.003004, 0.000000, 0.00
                    {0.998856, 0.000009, 0.000976, 0.000002, 0.000000, 0.000013, 0.00
                    {0.000178, 0.000446, 0.000326, 0.000033, 0.000193, 0.000071, 0.99
                    {0.000005, 0.000016, 0.000153, 0.000045, 0.004110, 0.000012, 0.00
                    {0.994188, 0.000003, 0.002584, 0.000005, 0.000005, 0.000100, 0.00
                    {0.000173, 0.990569, 0.000792, 0.000040, 0.001798, 0.000035, 0.00
                    {0.000000, 0.000537, 0.000008, 0.005080, 0.000046, 0.992910, 0.00
                    {0.000127, 0.000003, 0.000003, 0.000000, 0.001583, 0.000000, 0.00
                    {0.000001, 0.000012, 0.000072, 0.000020, 0.000000, 0.000000, 0.00
                    {0.000020, 0.000105, 0.001139, 0.901343, 0.002132, 0.083873, 0.00
                    {0.000002, 0.000048, 0.000019, 0.000000, 0.999347, 0.000002, 0.00
                    {0.000059, 0.001344, 0.612502, 0.002749, 0.000229, 0.000678, 0.00
                    {0.000586, 0.000740, 0.001625, 0.000007, 0.269341, 0.000076, 0.01
                    {0.009547, 0.018055, 0.283795, 0.071079, 0.426074, 0.082335, 0.03
                    {0.002506, 0.002545, 0.001148, 0.005659, 0.020416, 0.000112, 0.00
                    {0.001263, 0.001769, 0.000293, 0.000011, 0.000302, 0.881768, 0.11
                    {0.002904, 0.002909, 0.013421, 0.001461, 0.007519, 0.001251, 0.00
                    {0.000055, 0.001080, 0.893158, 0.000000, 0.104492, 0.000159, 0.00
                    {0.000344, 0.002693, 0.071184, 0.000262, 0.000001, 0.000003, 0.00
                    {0.001404, 0.009375, 0.002638, 0.229189, 0.000064, 0.000896, 0.00
                    {0.491140, 0.000125, 0.000024, 0.000302, 0.000038, 0.034947, 0.47

   vector top_k=y_true.ClassificationScore(y_scores,CLASSIFICATION_TOP_K_ACCURACY,1);
   Print("top 1 accuracy score = ",top_k);
   top_k=y_true.ClassificationScore(y_scores,CLASSIFICATION_TOP_K_ACCURACY,2);
   Print("top 2 accuracy score = ",top_k);
   vector y_true2={0, 1, 2, 2};
   matrix y_score2={{0.5, 0.2, 0.2},   // 0 is in top 2
                    {0.3, 0.4, 0.2},   // 1 is in top 2
                    {0.2, 0.4, 0.3},   // 2 is in top 2
                    {0.7, 0.2, 0.1}}; // 2 isn't in top 2
   top_k=y_true2.ClassificationScore(y_score2,CLASSIFICATION_TOP_K_ACCURACY,2);
   Print("top k = ",top_k);
   Print("");

   vector ap_micro=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISI
   Print("average precision score micro = ",ap_micro);
   vector ap_macro=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISI
   Print("average precision score macro = ",ap_macro);
   vector ap_weighted=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PREC
   Print("average precision score weighted = ",ap_weighted);
   vector ap_none=y_true.ClassificationScore(y_scores,CLASSIFICATION_AVERAGE_PRECISION
   Print("average precision score none = ",ap_none);
   Print("");

   vector area_micro=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERA
   Print("roc auc score micro = ",area_micro);
   vector area_macro=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERA
   Print("roc auc score macro = ",area_macro);
   vector area_weighted=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AV
   Print("roc auc score weighted = ",area_weighted);
   vector area_none=y_true.ClassificationScore(y_scores,CLASSIFICATION_ROC_AUC,AVERAG
   Print("roc auc score none = ",area_none);
   Print("");

//--- binary classification
   vector y_pred_bin={0,1,0,1,1,0,0,0,1};
   vector y_true_bin={1,0,0,0,1,0,1,1,1};
   vector y_score_true={0.3,0.7,0.1,0.6,0.9,0.0,0.4,0.2,0.8};
   matrix y_score1_bin(y_score_true.Size(),1);
   y_score1_bin.Col(y_score_true,0);
   matrix y_scores_bin={{0.7, 0.3},
                        {0.3, 0.7},
                        {0.9, 0.1},
                        {0.4, 0.6},
                        {0.1, 0.9},
                        {1.0, 0.0},
                        {0.6, 0.4},
                        {0.8, 0.2},
                        {0.2, 0.8}};

   vector ap=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_AVERAGE_PRECI
   Print("average precision score binary = ",ap);
   vector ap2=y_true_bin.ClassificationScore(y_score1_bin,CLASSIFICATION_AVERAGE_PREC
   Print("average precision score binary = ",ap2);
   vector ap3=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_AVERAGE_PREC
   Print("average precision score none = ",ap3);
   Print("");

   vector area=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_ROC_AUC,AVE
   Print("roc auc score binary = ",area);
   vector area2=y_true_bin.ClassificationScore(y_score1_bin,CLASSIFICATION_ROC_AUC,AV
   Print("roc auc score binary = ",area2);
   vector area3=y_true_bin.ClassificationScore(y_scores_bin,CLASSIFICATION_ROC_AUC,AV
   Print("roc auc score none = ",area3);


/*
  top 1 accuracy score = [0.6666666666666666]
  top 2 accuracy score = [1]
  top k = [0.75]

  average precision score micro = [0.8513333333333333]
  average precision score macro = [0.9326666666666666]
  average precision score weighted = [0.9333333333333333]
  average precision score none = [1,1,0.7,1,0.9266666666666666,0.8333333333333333,1,0

  roc auc score micro = [0.9839506172839506]
  roc auc score macro = [0.9892068783068803]
  roc auc score weighted = [0.9887354497354497]
  roc auc score none = [1,1,0.9506172839506173,1,0.984,0.9821428571428571,1,0.9753086

  average precision score binary = [0.7961904761904761]
  average precision score binary = [0.7961904761904761]
  average precision score none = [0.7678571428571428,0.7961904761904761]

  roc auc score binary = [0.7]
  roc auc score binary = [0.7]
  roc auc score none = [0.7,0.7]
*/
```

PrecisionRecall

Compute values to construct a precision-recall curve. Similarly to ClassificationScore, this method is applied to the vector of true values.

```
bool vector::PrecisionRecall(
   const matrix&                pred_scores,    // matrix containing the probability c
   const ENUM_ENUM_AVERAGE_MODE mode            // averaging mode
   matrix&                      precision,      // calculated precision values for eac
   matrix&                      recall,         // calculated recall values for each t
   matrix&                      thresholds,     // threshold values sorted in descendi
   );
```

Parameters

pred_scores

[in] A matrix containing a set of horizontal vectors with probabilities for each class. The number of matrix rows must correspond to the size of the vector of true values.

mode

[in] Averaging mode from the ENUM_AVERAGE_MODE enumeration. Only AVERAGE_NONE, AVERAGE_BINARY and AVERAGE_MICRO are used.

precision

[out] A matrix with calculated precision curve values. If no averaging is applied (AVERAGE_NONE), the number of rows in the matrix corresponds to the number of model classes. The number of columns corresponds to the size of the vector of true values (or the number of rows in the probability distribution matrix pred_score). In the case of microaveraging, the number of rows in the matrix corresponds to the total number of threshold values, excluding duplicates.

recall

[out] A matrix with calculated recall curve values.

threshold

[out] Threshold matrix obtained by sorting the probability matrix

Note

See notes for the ClassificationScore method.

Example

An example of collecting statistics from the mnist.onnx model (99% accuracy).

```
//--- data for classification metrics
   vectorf y_true(images);
   vectorf y_pred(images);
   matrixf y_scores(images,10);
//--- input-output
   matrixf image(28,28);
   vectorf result(10);

//--- testing
   for(int test=0; test<images; test++)
     {
      image=test_data[test].image;
      if(!OnnxRun(model,ONNX_DEFAULT,image,result))
        {
         Print("OnnxRun error ",GetLastError());
         break;
        }
      result.Activation(result,AF_SOFTMAX);
      //--- collect data
      y_true[test]=(float)test_data[test].label;
      y_pred[test]=(float)result.ArgMax();
      y_scores.Row(result,test);
     }     }
```

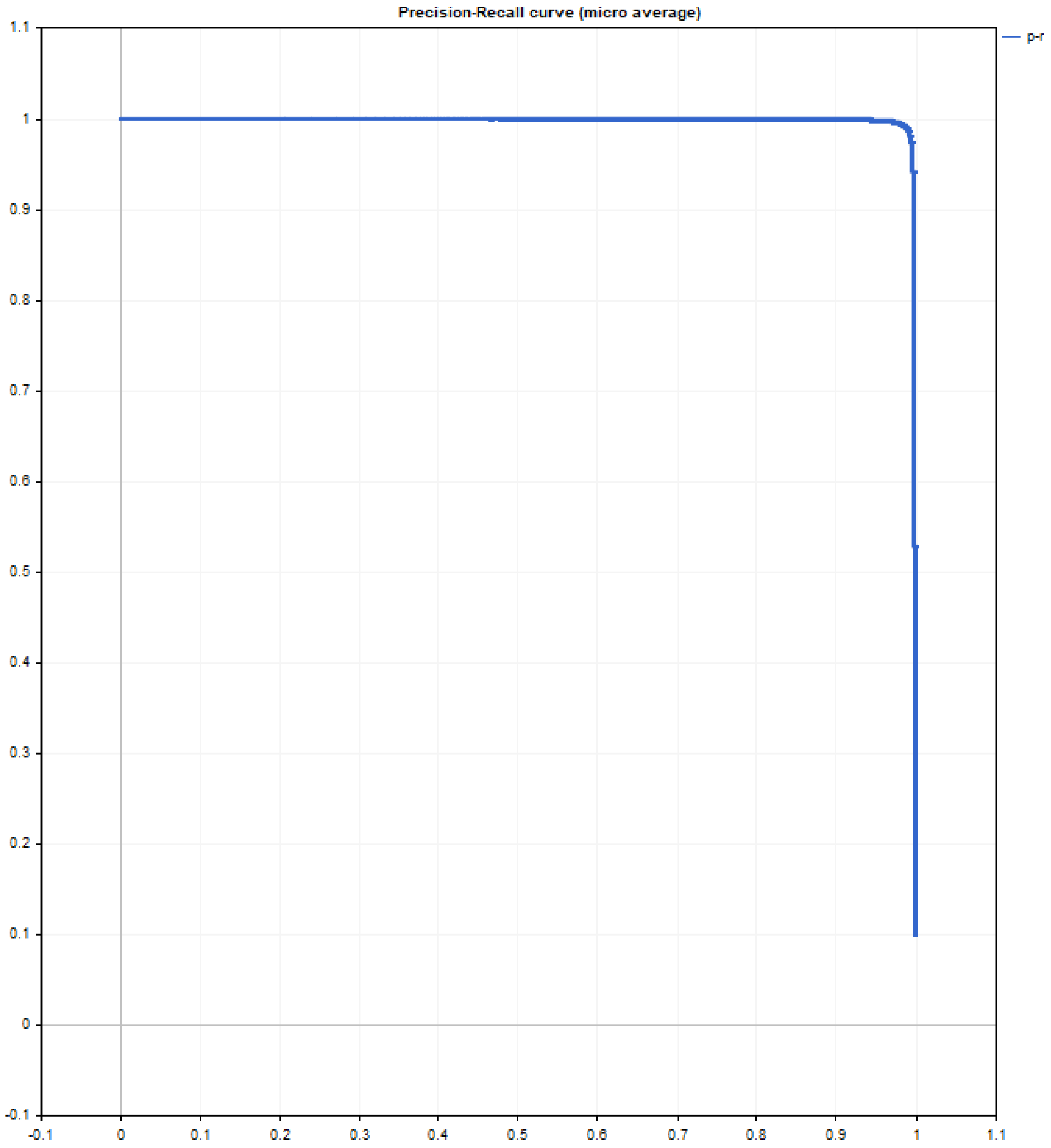
Accuracy calculation

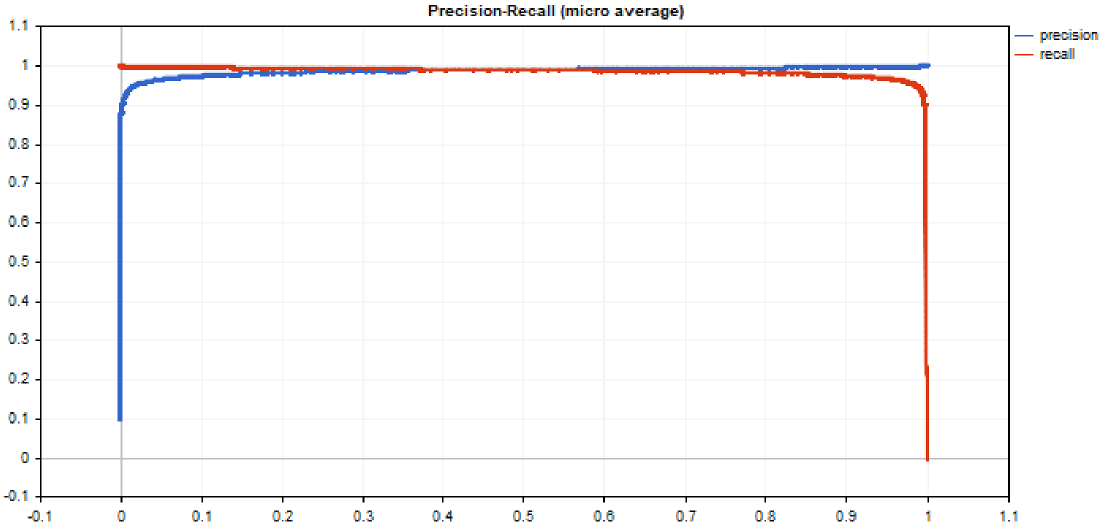
```
   vectorf accuracy=y_pred.ClassificationMetric(y_true,CLASSIFICATION_ACCURACY);
   PrintFormat("accuracy=%f",accuracy[0]);

accuracy=0.989000
```

An example of plotting precision-recall graphs, where precision values are plotted on the y-axis and recall values are plotted on the x-axis. Also precision and recall graphs are plotted separately, with threshold values plotted on the x-axis

```
   if(y_true.PrecisionRecall(y_scores,AVERAGE_MICRO,mat_precision,mat_recall,mat_thres
     {
      double precision[],recall[],thres[];
      ArrayResize(precision,mat_thres.Cols());
      ArrayResize(recall,mat_thres.Cols());
      ArrayResize(thres,mat_thres.Cols());

      for(uint i=0; i<thres.Size(); i++)
        {
         precision[i]=mat_precision[0][i];
         recall[i]=mat_recall[0][i];
         thres[i]=mat_thres[0][i];
        }
      thres[0]=thres[1]+0.001;

      PlotCurve("Precision-Recall curve (micro average)","p-r","",recall,precision);
      Plot2Curves("Precision-Recall (micro average)","precision","recall",thres,precis
     }
```

Resulting curves:




Precision-Recall (micro average)
precision
recall
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1

ReceiverOperatingCharacteristic

Compute values to construct the Receiver Operating Characteristic (ROC) curve. Similarly to ClassificationScore, this method is applied to the vector of true values.

```
bool vector::ReceiverOperatingCharacteristic(
   const matrix&                pred_scores,    // matrix containing the probability d
   const ENUM_ENUM_AVERAGE_MODE mode            // averaging mode
   matrix&                      fpr,            // calculated false positive rate valu
   matrix&                      tpr,            // calculated true positive rate value
   matrix&                      thresholds,     // threshold values sorted in descendi
   );
```

Parameters

pred_scores

[in] A matrix containing a set of horizontal vectors with probabilities for each class. The number of matrix rows must correspond to the size of the vector of true values.

mode

[in] Averaging mode from the ENUM_AVERAGE_MODE enumeration. Only AVERAGE_NONE, AVERAGE_BINARY and AVERAGE_MICRO are used.

fpr

[out] A matrix with calculated values of the false positive rate curve. If no averaging is applied (AVERAGE_NONE), the number of rows in the matrix corresponds to the number of model classes. The number of columns corresponds to the size of the vector of true values (or the number of rows in the probability distribution matrix pred_score). In the case of microaveraging, the number of rows in the matrix corresponds to the total number of threshold values, excluding duplicates.

tpr

[out] A matrix with calculated values of the true positive rate curve.

threshold

[out] Threshold matrix obtained by sorting the probability matrix

Note

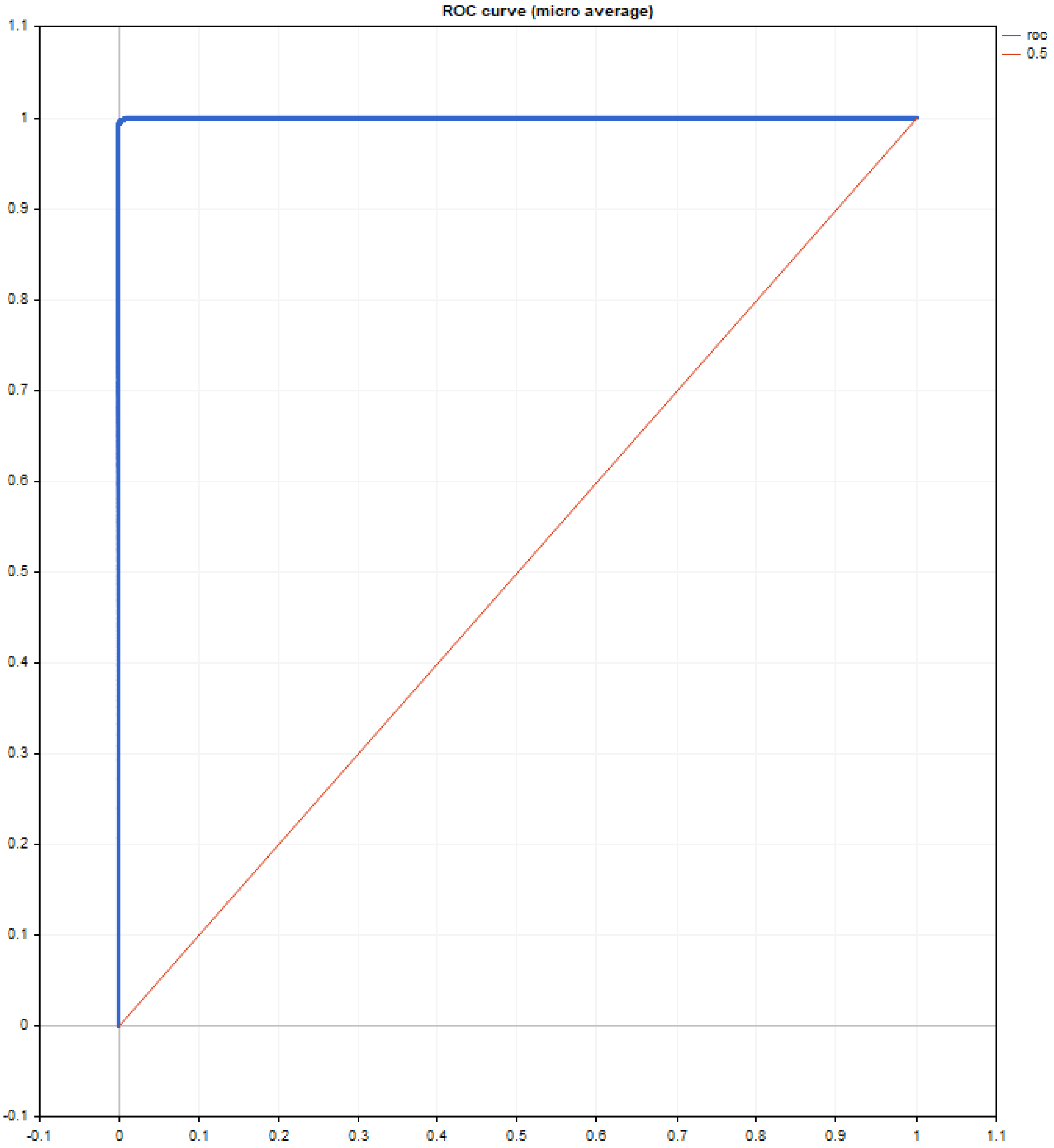
See notes for the ClassificationScore method.

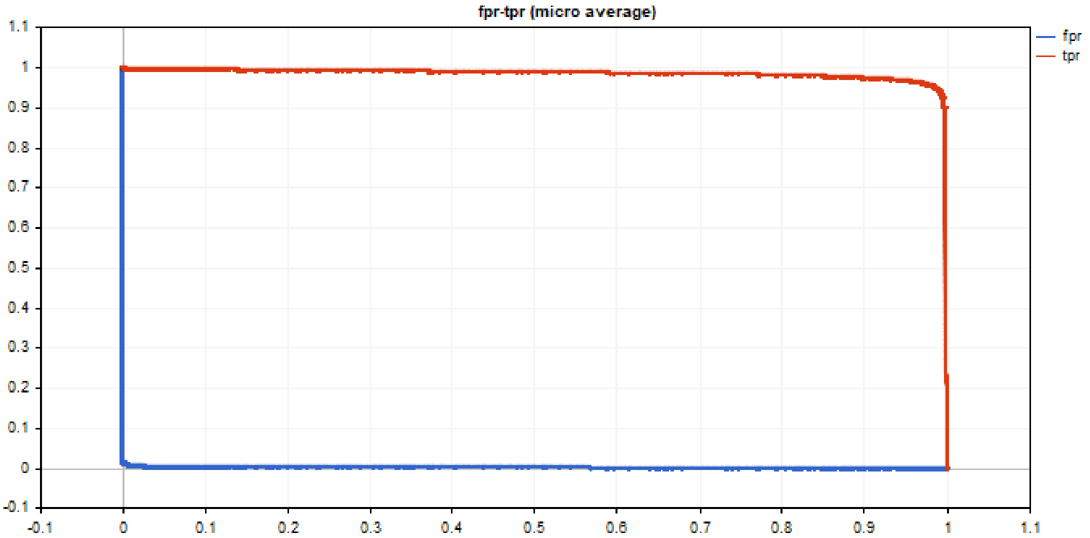
Example

An example of plotting ROC graphs, where tpr values are plotted on the y-axis and fpr values are plotted on the x-axis. Also fpr and tpr graphs are plotted separately, with threshold values plotted on the x-axis

```
   matrixf mat_thres;
   matrixf mat_fpr;
   matrixf mat_tpr;
   if(y_true.ReceiverOperatingCharacteristic(y_scores,AVERAGE_MICRO,mat_fpr,mat_tpr,ma
     {
      double fpr[],tpr[],thres[];
      ArrayResize(fpr,mat_thres.Cols());
      ArrayResize(tpr,mat_thres.Cols());
      ArrayResize(thres,mat_thres.Cols());

      for(uint i=0; i<fpr.Size(); i++)
        {
         fpr[i]=mat_fpr[0][i];
         tpr[i]=mat_tpr[0][i];
         thres[i]=mat_thres[0][i];
        }
      thres[0]=thres[1]+0.001;

      PlotCurve("ROC curve (micro average)","roc","0.5",fpr,tpr);
      Plot2Curves("fpr-tpr (micro average)","fpr","tpr",thres,fpr,tpr);
     }
```

Resulting curves:


ROC curve (micro average)
roc
0.5
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1


fpr-tpr (micro average)
fpr
tpr
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1

The graph output code is simple and based on the <Graphics/Graphic.mqh> standard library.

The examples use the data of the mnist.onnx model. The code is presented in the PrecisionRecall method description.

ROC AUC is close to ideal.

```
roc auc score micro = [0.99991]
```

OpenBLAS Methods

OpenBLAS is a high-performance open-source linear algebra library that implements BLAS (Basic Linear Algebra Subprograms) and some LAPACK functions. OpenBLAS is designed to improve computational performance, particularly in matrix and vector operations, which are often used in scientific and engineering tasks such as machine learning, numerical methods, and simulations.

Key features of OpenBLAS:

- Multithreading support: OpenBLAS can efficiently use multiple processor cores for parallel computations, significantly accelerating operations on multiprocessor systems.
- Optimization for processor architectures: OpenBLAS includes optimized builds for various processors such as Intel, AMD, ARM and others. The library automatically detects processor characteristics and selects the most suitable function implementations.
- Extensive BLAS operation support: OpenBLAS implements core BLAS functions, including vector operations (e.g., vector addition and dot product), matrix operations (multiplication), and vector-matrix operations.
- LAPACK compatibility: The library supports LAPACK (Linear Algebra PACKage) functions for more complex linear algebra operations, such as solving systems of linear equations, calculating matrix eigenvalues, and others.
- High performance: Compared to other BLAS libraries, OpenBLAS often shows better results due to hand-optimization for specific processor architectures.

Applications

OpenBLAS is widely used in applications involving numerical computations:

- Training neural networks and other machine learning tasks.
- Scientific computing (e.g. modeling of physical processes).
- Processing and analyzing large amounts of data.

The library is integrated into many popular scientific software packages such as NumPy, SciPy, and TensorFlow, which rely on high-performance linear algebra operations.

OpenBLAS is an excellent choice for those seeking an open-source solution for high-performance computing, particularly when working with large matrices and vectors.

Function	Action
SingularValueDecompositionDC	Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD).
SingularValueDecompositionQR	Singular Value Decomposition, QR algorithm. This algorithm is considered a classical SVD algorithm (lapack function GESVD).

Function	Action
SingularValueDecompositionQRPivot	Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ).
SingularValueDecompositionBisect	Singular Value Decomposition, bisection algorithm (lapack function GESVDX).
SingularValueDecompositionJacobiHigh	Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV).
SingularValueDecompositionJacobiLow	Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases.
SingularValueDecompositionBidiagDC	Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSVDX).
SingularValueDecompositionBidiagBisect	Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX).
EigenSolver	Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV).
EigenSolver2	Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV).
EigenSolverX	Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX).
EigenSolverShur	Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition.
EigenSymmetricDC	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD).
EigenSymmetricQR	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex

Function	Action
	conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV).
EigenSymmetricRobust	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR).
EigenSymmetricBisect	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX).
SingularSpectrumAnalysisSpectrum	A method function for calculating the relative contributions of spectral components based on their eigenvalues.
SingularSpectrumAnalysisForecast	A method function for calculating reconstructed and predicted data using spectral components of the input time series.
SingularSpectrumAnalysisReconstructComponents	A method function for calculating reconstructed components of the input time series and their contributions.
SingularSpectrumAnalysisReconstructSeries	A method function for calculating the reconstructed time series using the first component_count components.

Singular value decomposition

This section features functions for decomposing a matrix into three components: orthogonal matrices and a diagonal matrix of singular values. SVD is applied to solve various linear algebra problems such as data dimensionality reduction, image compression, solving systems of equations, and data analysis and optimization. The main functions allow you to compute singular values and vectors, reconstruct matrices, and approximate matrices with reduced rank accuracy.

Function	Action
SingularValueDecompositionDC	Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD).
SingularValueDecompositionQR	Singular Value Decomposition, QR algorithm. This algorithm is considered a classical SVD algorithm (lapack function GESVD).
SingularValueDecompositionQRPivot	Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ).
SingularValueDecompositionBisect	Singular Value Decomposition, bisection algorithm (lapack function GESVDX).
SingularValueDecompositionJacobiHigh	Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV).
SingularValueDecompositionJacobiLow	Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases.
SingularValueDecompositionBidiagDC	Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSVDX).
SingularValueDecompositionBidiagBisect	Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX).
SingularValueDecompositionBidiagQR	Computes the singular value decomposition of a general matrix that has been reduced to bidiagonal form by the method ReduceToBidiagonal. Lapack function BDSQR.

SingularValueDecompositionDC

Singular Value Decomposition, "divide-and-conquer" algorithm. This algorithm is considered the fastest among other SVD algorithms (lapack function GESDD).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z       jobz,      // how to computed
   vector&          S,         // vector of computed singular values
   matrix&          U,         // matrix of computed left vectors U
   matrix&          VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z       jobz,      // how to computed
   vectorf&         S,         // vector of computed singular values
   matrixf&         U,         // matrix of computed left vectors U
   matrixf&         VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z       jobz,      // how to computed
   vector&          S,         // vector of computed singular values
   matrixc&         U,         // matrix of computed left vectors U
   matrixc&         VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionDC(
   ENUM_SVD_Z       jobz,                   // how to computed
   vectorf&         singular_values,        // vector of computed singular values
   matrixcf&        U,                      // matrix of computed left vectors U
   matrixcf&        VT                      // transposed matrix of right vectors VT
   );
```

Parameters

jobz

[in] ENUM_SVD_Z enumeration value which determines the method for computing left and singular eigenvectors.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

VT

[out] Matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobz parameter.

When jobv is set to SVDZ_N, the left and right vectors are not computed. Only singular values are computed.

When jobv is set to SVDZ_A, the full matrices of the U and VT vectors are computed.

When the value is SVDZ_S, truncated matrices of vectors U and VT are computed.

ENUM_SVD_Z

An enumeration defining the way to compute left and right singular vectors.

ID	Description
SVDZ_N	Columns U or rows VT are not computed
SVDZ_A	All M columns of U or all N columns of VT are returned in arrays U and VT
SVDZ_S	The first min(M,N) columns of U or the first min(M,N) columns of VT are returned in arrays U and VT

See also

SingularValueDecompositionQR, SingularValueDecompositionQRPivot

SingularValueDecompositionQR

Singular Value Decomposition, QR algorithm. Considered a classical SVD algorithm (lapack function GESVD).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z       jobu,      // how to compute left vectors
   ENUM_SVD_Z       jobv,      // how to compute right vectors
   vector&          S,         // vector of computed singular values
   matrix&          U,         // matrix of computed left vectors U
   matrix&          VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z       jobu,      // how to compute left vectors
   ENUM_SVD_Z       jobv,      // how to compute right vectors
   vectorf&         S,         // vector of computed singular values
   matrixf&         U,         // matrix of computed left vectors U
   matrixf&         VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z       jobu,      // how to compute left vectors
   ENUM_SVD_Z       jobv,      // how to compute right vectors
   vector           S,         // vector of computed singular values
   matrixc          U,         // matrix of computed left vectors U
   matrixc          VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionQR(
   ENUM_SVD_Z       jobu,                     // how to compute left vectors
   ENUM_SVD_Z       jobv,                     // how to compute right vectors
   vectorf&         singular_values,          // vector of computed singular values
   matrixcf&        u,                        // matrix of computed left vectors U
   matrixcf&        vt                        // transposed matrix of right vectors VT
   );
```

Parameters

jobu

[in] ENUM_SVD_Z enumeration value defining how the left singular vectors should be computed.

jobv

[in] ENUM_SVD_Z enumeration value defining how the right singular vectors should be computed.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

VT

[out] Matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobu and jobv parameters.

When set to SVDZ_N, the left (jobu) and right (jobv) vectors are not computed. Singular values are always computed.

When set to SVDZ_A, the full matrices of vectors U (jobu) or VT (jobv) are computed.

With the value SVDZ_S, truncated matrices of vectors U (jobu) or VT (jobv) are computed.

ENUM_SVD_Z

An enumeration defining the way to compute left and right singular vectors.

ID	Description
SVDZ_N	Columns U or rows VT are not computed
SVDZ_A	All M columns of U or all N columns of VT are returned in arrays U and VT
SVDZ_S	The first min(M,N) columns of U or the first min(M,N) columns of VT are returned in arrays U and VT

See also

SingularValueDecompositionDC, SingularValueDecompositionQRPivot

SingularValueDecompositionQRPivot

Singular Value Decomposition, QR with pivoting algorithm (lapack function GESVDQ).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A    joba,      // computation accuracy level
   ENUM_SVDQRP_P    jobp,      // use row reversal to compute
   ENUM_SVDQRP_R    jobr,      // use triangular matrix R to compute
   ENUM_SVDQRP_U    jobu,      // how to compute left vectors
   ENUM_SVDQRP_V    jobv,      // how to compute right vectors
   vector&          S,         // vector of computed singular values
   matrix&          U,         // matrix of computed left vectors U
   matrix&          VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A    joba,      // computation accuracy level
   ENUM_SVDQRP_P    jobp,      // use row reversal to compute
   ENUM_SVDQRP_R    jobr,      // use triangular matrix R to compute
   ENUM_SVDQRP_U    jobu,      // how to compute left vectors
   ENUM_SVDQRP_V    jobv,      // how to compute right vectors
   vectorf&         S,         // vector of computed singular values
   matrixf&         U,         // matrix of computed left vectors U
   matrixf&         VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A    joba,      // computation accuracy level
   ENUM_SVDQRP_P    jobp,      // use row reversal to compute
   ENUM_SVDQRP_R    jobr,      // use triangular matrix R to compute
   ENUM_SVDQRP_U    jobu,      // how to compute left vectors
   ENUM_SVDQRP_V    jobv,      // how to compute right vectors
   vector&          S,         // vector of computed singular values
   matrixc&         U,         // matrix of computed left vectors U
   matrixc&         VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionQRPivot(
   ENUM_SVDQRP_A    joba,                   // computation accuracy level
   ENUM_SVDQRP_P    jobp,                   // use row reversal to compute
   ENUM_SVDQRP_R    jobr,                   // use triangular matrix R to compute
   ENUM_SVDQRP_U    jobu,                   // how to compute left vectors
   ENUM_SVDQRP_V    jobv,                    // how to compute right vectors
   vectorf&         singular_values,         // vector of computed singular values
   matrixcf&        u,                       // matrix of computed left vectors U
   matrixcf&        vt                       // transposed matrix of right vectors VT
   );
```

Parameters

joba

[in] ENUM_SVDQRP_A enumeration value defining the accuracy level of the SVD computation.

jobp

[in] ENUM_SVDQRP_P enumeration value defining the use of row reversal during the computation process.

jobr

[in] ENUM_SVDQRP_R enumeration value defining whether to transpose the triangular matrix R obtained as a result of the initial QR factorization.

jobu

[in] ENUM_SVDQRP_U enumeration value that determines how the left singular vectors should be computed.

jobv

[in] ENUM_SVDQRP_V enumeration value defining how the right singular vectors should be computed.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

VT

[out] Matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The number of matrix rows must not be less than the number of columns.

If both left and right singular vectors are computed, jobv must be set to SVDQRPV_R when jobu=SVDQRPU_R.

ENUM_SVDQRP_A

An enumeration that specifies the level of accuracy of the SVD computation.

ID	Description
SVDQRP_A	The requested accuracy corresponds to the inverse error limited by epsilon. This is an aggressive level of truncation.
SVDQRPA_M	Similar to SVDQRP_A, but the truncation is softer. This is the average level of truncation.
SVDQRPA_H	High accuracy is required.

ENUM_SVDQRP_P

An enumeration that specifies whether to use row reversal during calculation.

ID	Description
SVDQRPP_P	The rows of A are ordered in descending order. Recommended for numerical reliability.
SVDQRPP_N	No row reversal.

ENUM_SVDQRP_R

An enumeration that specifies whether to transpose the triangular matrix R obtained as a result of the initial QR factorization.

ID	Description
SVDQRPR_T	After the initial rotated QR factorization, GESVD is applied to the transpose R**T of the calculated triangular factor R.
SVDQRPR_N	The triangular factor R is given as input to GESVD.

ENUM_SVDQRP_U

An enumeration defining how left singular vectors should be computed.

ID	Description
SVDQRPU_A	All M left singular vectors are computed.
SVDQRPU_S	The min(M,N) left singular vectors are computed.
SVDQRPU_R	The numerical rank NUMRANK is determined and only the NUMRANK of the left singular vectors is computed.
SVDQRPU_F	N left singular vectors are returned.
SVDQRPU_N	Left singular vectors are not computed.

ENUM_SVDQRP_V

An enumeration defining how right singular vectors should be computed.

ID	Description
SVDQRPV_A	All N right singular vectors are computed.
SVDQRPV_R	The numerical rank NUMRANK is defined and only the NUMRANK of right singular vectors are computed.
SVDQRPV_N	Right singular vectors are not computed.

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

SingularValueDecompositionBisect

Singular Value Decomposition, bisection algorithm (lapack function GESVDX).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS    jobv,      // computation accuracy level
   ENUM_BLAS_RANGE     range,     // subset of computable singular values
   double              lower,     // lower limit of the subset
   double              upper,     // upper limit of the subset
   vector&             S,         // vector of computed singular values
   matrix&             U,         // matrix of computed left vectors U
   matrix&             VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS    jobv,      // computation accuracy level
   ENUM_BLAS_RANGE     range,     // subset of computable singular values
   double              lower,     // lower limit of the subset
   double              upper,     // upper limit of the subset
   vectorf&            S,         // vector of computed singular values
   matrixf&            U,         // matrix of computed left vectors U
   matrixf&            VT         // ransposed matrix of right vectors VT
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS    jobv,      // computation accuracy level
   ENUM_BLAS_RANGE     range,     // subset of computable singular values
   double              lower,     // lower limit of the subset
   double              upper,     // upper limit of the subset
   vector&             S,         // vector of computed singular values
   matrixc&            U,         // matrix of computed left vectors U
   matrixc&            VT         // transposed matrix of right vectors VT
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionBisect(
   ENUM_SVD_VECTORS    jobv,                // computation accuracy level
   ENUM_BLAS_RANGE     range,               // subset of computable singular values
   double              lower,               // lower limit of the subset
   double              upper,               // upper limit of the subset
   vectorf&            singular_values,     // vector of computed singular values
   matrixcf&           u,                   // matrix of computed left vectors U
   matrixcf&           vt                   // transposed matrix of right vectors VT
   );
```

Parameters

jobv

[in] ENUM_SVD_VECTORS enumeration value which determines the method for computing left and singular eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

lower

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

upper

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

VT

[out] Matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobuv* and *range* parameters.

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

ENUM_SVD_VECTORS

An enumeration defining the way to compute left and right singular vectors.

ID	Description
SVDVECTORS_N	Only singular values are computed, without vectors.
SVDVECTORS_U	Left singular vectors are computed.
SVDVECTORS_V	Right singular vectors are computed.
SVDVECTORS_UV	Left and right singular vectors are computed.

ENUM_BLAS_RANGE

An enumeration defining a subset of computable singular values and vectors.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	Singular or eigenvalues from IL to IU will be found

See also

SingularValueDecompositionDC, SingularValueDecompositionQR, SingularValueDecompositionQRPivot

SingularValueDecompositionJacobiHigh

Singular Value Decomposition, Jacobi high level algorithm (lapack function GEJSV).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A    joba,            // computation accuracy level
   ENUM_SVDJH_U    jobu,            // how to compute left vectors
   ENUM_SVDJH_V    jobv,            // how to compute right vectors
   ENUM_SVDJH_R    jobr,            // define range of computable singular values
   ENUM_SVDJH_T    jobt,            // define whether to transpose when computing a squ
   ENUM_SVDJH_P    jobp,            // the possibility of structured perturbations to r
   vector&         S,               // vector of computed singular values
   matrix&         U,               // matrix of computed left vectors U
   matrix&         V,               // matrix of computed left vectors V
   vector&         work_results     // additional computation results
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A    joba,            // computation accuracy level
   ENUM_SVDJH_U    jobu,            // how to compute left vectors
   ENUM_SVDJH_V    jobv,            // how to compute right vectors
   ENUM_SVDJH_R    jobr,            // define range of computable singular values
   ENUM_SVDJH_T    jobt,            // define whether to transpose when computing a squ
   ENUM_SVDJH_P    jobp,            // the possibility of structured perturbations to r
   vectorf&        S,               // vector of computed singular values
   matrixf&        U,               // U matrix of computed left vectors
   matrixf&        V,               // V matrix of computed left vectors
   vector&         work_results     // additional computation results
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A    joba,            // computation accuracy level
   ENUM_SVDJH_U    jobu,            // how to compute left vectors
   ENUM_SVDJH_V    jobv,            // how to compute right vectors
   ENUM_SVDJH_R    jobr,            // define range of computable singular values
   ENUM_SVDJH_T    jobt,            // define whether to transpose when computing a squ
   ENUM_SVDJH_P    jobp,            // the possibility of structured perturbations to r
   vectorc&        S,               // vector of computed singular values
   matrixc&        U,               // U matrix of computed left vectors
   matrixc&        V,               // V matrix of computed left vectors
   vector&         work_results     // additional computation results
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionJacobiHigh(
   ENUM_SVDJH_A     joba,                       // computation accuracy level
   ENUM_SVDJH_U     jobu,                       // how to compute left vectors
   ENUM_SVDJH_V     jobv,                       // how to compute right vectors
   ENUM_SVDJH_R     jobr,                       // define range of computable singular valu
   ENUM_SVDJH_T     jobt,                       // define whether to transpose when computi
   ENUM_SVDJH_P     jobp,                       // the possibility of structured perturbati
   vectorf&         singular_values,            // vector of computed singular values S
   matrixcf&        u,                          // matrix of computed left vectors U
   matrixcf&        v,                          // matrix of computed left vectors V
   vectorf&         work_results                // additional computation results
   );
```

Parameters

joba

[in] ENUM_SVDJH_A enumeration value determining the accuracy level of the SVD computation.

jobu

[in] ENUM_SVDJH_U enumeration value that determines how the left singular vectors should be computed.

jobv

[in] ENUM_SVDJH_V enumeration value that determines how the right singular vectors should be computed.

jobr

[in] ENUM_SVDJH_R enumeration value defining the range of computable values

jobt

[in] ENUM_SVDJH_T enumeration value defining whether to transpose the matrix of it is square. If the matrix is non-square, this parameter is ignored.

jobp

[in] ENUM_SVDJH_P enumeration value defining the possibility of structured perturbations to remove denormalized values.

S

[out] Vector of singular values.

For work(1)/work(2) = one: the singular values of A. During the computation S contains Euclidean column norms of the iterated matrices in the array a.

For work(1)≠work(2): the singular values of A are (work(1)/work(2)) * S(1:n). This factored form is used if sigma_max(A) overflows or if small singular values have been saved from underflow by scaling the input matrix A.

jobr = 'R', some of the singular values may be returned as exact zeros obtained by 'setting to zero' because they are below the numerical rank threshold or are denormalized numbers.

U

[out] Matrix of left singular vectors.

V

[out] Matrix of right singular vectors.

work_results

[out] Vector consisting of 7 statistics obtained as a result of the computation.

work(1) = scale = work(2)/work(1) is the scaling factor such that scale*sva(1:n) are the computed singular values of A. See the description of S.

work(2) = see the description of work(1).

work(3) = sconda is an estimate for the condition number of column equilibrated A. If joba = 'E' or 'G', sconda is an estimate of sqrt(||(R**t * R)**(-1)||_1). It is computed using ?pocon. It holds n**(-1/4) * sconda ≤ ||R**(-1)||_2 ≤ n**(1/4) * sconda, where R is the triangular factor from the QRF of A. However, if R is truncated and the numerical rank is determined to be strictly smaller than n, sconda is returned as -1, indicating that the smallest singular values might be lost.

If full SVD is needed, the following two condition numbers are useful for the analysis of the algorithm. They are provied for a user who is familiar with the details of the method.

work(4) = an estimate of the scaled condition number of the triangular factor in the first QR factorization.

work(5) = an estimate of the scaled condition number of the triangular factor in the second QR factorization.

The following two parameters are computed if jobt = 'T'. They are provided for a user who is familiar with the details of the method.

work(6) = the entropy of A**t*A :: this is the Shannon entropy of diag(A**t*A) / Trace(A**t*A) taken as point in the probability simplex.

work(7) = the entropy of A*A**t.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The number of matrix rows must not be less than the number of columns.

ENUM_SVDJH_A

An enumeration that specifies the level of accuracy of the SVD computation.

ID	Description
SVDJHA_C	Computation as with 'C' with an additional estimate of the condition number. It provides a realistic error bound.
SVDJHA_E	Computation as with SVDJHA_C' with an additional estimate of the condition number. It provides a realistic error bound.

ID	Description
SVDJHA_F	Higher accuracy than the SVDJHA_C option.
SVDJHA_G	Computation as with SVDJHA_F with an additional estimate of the condition number.
SVDJHA_A	Small singular values are the noise and the matrix is treated as numerically rank deficient.
SVDJHA_R	Similar as in SVDJHA_A, but more accuracy.

ENUM_SVDJH_U

An enumeration defining how left singular vectors should be computed.

ID	Description
SVDJHU_U	N columns of U are returned in the array U.
SVDJHU_F	Full set of M left singular vectors is returned in the array U.
SVDJHU_N	U is not computed.

ENUM_SVDJH_V

An enumeration defining how right singular vectors should be computed.

ID	Description
SVDJHV_V	N columns of V are returned in the array V
SVDJHV_J	N columns of V are returned in the array V, but they are computed as the product of Jacobi rotations
SVDJHV_N	V is not computed

ENUM_SVDJH_R

An enumeration that defines the range of values to be computed.

ID	Description
SVDJHR_N	Do not kill small columns of c*A.
SVDJHR_R	RESTRICTED range for sigma(c*A). This option is recommended.

ENUM_SVDJH_T

An enumeration that specifies whether a matrix should be transposed if it is square.

ID	Description
SVDJHT_T	Transpose if entropy test indicates possibly faster convergence of Jacobi process.
SVDJHT_N	Do not use transposition. Do not speculate.

ENUM_SVDJH_P

An enumeration that specifies the possibility of structured perturbations to remove denormalized values.

ID	Description
SVDJHP_P	Introduce perturbation.
SVDJHP_N	Do not perturb.

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

SingularValueDecompositionJacobiLow

Singular Value Decomposition, Jacobi low level algorithm (lapack function GESVJ). The method computes small singular values and their singular vectors with much greater accuracy than other SVD routines in certain cases.

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vector&        S,               // vector of computed singular values
   matrix&        U,               // matrix of computed left vectors U
   matrix&        V,               // matrix of computed left vectors V
   vector&        work_results     // additional computation results
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vectorf&       S,               // vector of computed singular values
   matrixf&       U,               // U matrix of computed left vectors
   matrixf&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U   jobu,            // how to compute left vectors
   ENUM_SVDJH_V   jobv,            // how to compute right vectors
   double         ctol,            // threshold for convergence if jobu='C'
   ulong          mv,              // number of first rows of matrix V if jobv='A'
   vector&        S,               // vector of computed singular values
   matrixc&       U,               // U matrix of computed left vectors
   matrixc&       V,               // V matrix of computed left vectors
   vector&        work_results     // additional computation results
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionJacobiLow(
   ENUM_SVDJH_U    jobu,                    // how to compute left vectors
   ENUM_SVDJH_V    jobv,                    // how to compute right vectors
   double          ctol,                 // threshold for convergence if jobu='C'
   ulong           mv,                   // number of first rows of matrix V if jobv
   vectorf&        singular_values,      // vector of computed singular values S
   matrixcf&       u,                    // matrix of computed left vectors U
   matrixcf&       v,                    // matrix of computed left vectors V
   vectorf&        work_results          // additional computation results
   );
```

Parameters

jobu

[in] [ENUM_SVDJL_U](#) enumeration value that determines how the left singular vectors should be computed.

jobv

[in] [ENUM_SVDJL_V](#) enumeration value defining how the right singular vectors should be computed.

ctol

[in] Convergence threshold if jobu=SVDJLU_C. For other values of 'jobu' the parameter is ignored.

mv

[in] Number of rows of matrix V to be computed if jobv=SVDJLV_A. For other values of 'jobv' the parameter is ignored.

S

[out] Vector of singular values.

Depending on the value scale = work(1), where scale is the scaling factor:

if scale = 1, S(1:n) contains the computed singular values of a. During the computation, sva contains the Euclidean column norms of the iterated matrices in the array a.

if scale ≠ 1, the singular values of a are scale*S(1:n), and this factored representation is due to the fact that some of the singular values of a might underflow or overflow.

U

[out] Matrix of left singular vectors.

V

[out] Matrix of right singular vectors (non-transposed).

work_results

[out] Vector consisting of 7 statistics obtained as a result of the computation.

work(1) = scale is the scaling factor such that scale*S(1:n) are the computed singular values of A. See the description of S).

work(2) is the number of the computed nonzero singular value.

work(3) is the number of the computed singular values that are larger than the underflow threshold.

work(4) is the number of sweeps of Jacobi rotations needed for numerical convergence.

work(5) = max_{i.NE.j} |COS(A(:,i),A(:,j))| in the last sweep. This is useful information in cases when ?gesvj did not converge, as it can be used to estimate whether the output is still useful and for post festum analysis.

work(6) is the largest absolute value over all sines of the Jacobi rotation angles in the last sweep. It can be useful in a post festum analysis.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The number of matrix rows must not be less than the number of columns.

ENUM_SVDJL_U

An enumeration defining how left singular vectors should be computed.

ID	Description
SVDJLU_U	The left singular vectors corresponding to the nonzero singular values are computed and returned in the leading columns of A
SVDJLU_C	Analogous to SVDJLU_U, except that user can control the level of numerical orthogonality of the computed left singular vectors.
SVDJLU_N	The matrix U is not computed.

ENUM_SVDJL_V

An enumeration defining how right singular vectors should be computed.

ID	Description
SVDJLV_V	The matrix V is computed
SVDJLV_A	The Jacobi rotations are applied to the MV-by-N matrix V.
SVDJLV_N	The matrix V is not computed.

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

SingularValueDecompositionBidiagDC

Singular Value Decomposition, divide-and-conquer algorithm for bidiagonal matrices (lapack function BDSDC).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagDC(
   ENUM_SVDBIDIAG_Z     jobz,        // how to compute left vectors
   ENUM_BLAS_RANGE      range,       // subset of computed singular values
   double               lower,       // lower limit of the subset
   double               upper,       // upper limit of the subset
   vector&              S,           // vector of computed singular values
   matrix&              U,           // U matrix of computed left vectors
   matrix&              VT           // VT transposed matrix of right vectors
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagDC(
   ENUM_SVDBIDIAG_Z     jobz,        // how to compute left vectors
   ENUM_BLAS_RANGE      range,       // subset of computed singular values
   double               lower,       // lower limit of the subset
   double               upper,       // upper limit of the subset
   vectorf&             S,           // vector of computed singular values
   matrixf&             U,           // U matrix of computed left vectors
   matrixf&             VT           // VT transposed matrix of right vectors
   );
```

Parameters

jobz

[in] ENUM_SVDBIDIAG_Z enumeration value that determines how the left singular vectors should be computed.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

lower

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

upper

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

V

[out] Transposed matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the jobz and range parameters.

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

A bidiagonal matrix is a square matrix with non-zero main diagonal and one of the sub-diagonals.

Upper bidiagonal matrix

```
[[x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x],
 [0, 0, 0, 0, x]]
```

Lower bidiagonal matrix

```
[[x, 0, 0, 0, 0],
 [x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x]]
```

ENUM_SVDBIDIAG_Z

An enumeration defining how left singular vectors should be computed.

ID	Description
SVDJOBZ_V	Compute singular values and singular vectors.
SVDJOBZ_N	Compute singular values only.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

SingularValueDecompositionBidiagBisect

Singular Value Decomposition, bisection algorithm for bidiagonal matrices (lapack function BDSVDX).

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagBisect(
   ENUM_SVDBIDIAG_Z     jobz,         // how to compute left vectors
   ENUM_BLAS_RANGE      range,        // subset of computed singular values
   double               lower,        // lower limit of the subset
   double               upper,        // upper limit of the subset
   vector&              S,            // vector of computed singular values
   matrix&              U,            // U matrix of computed left vectors
   matrix&              VT            // VT transposed matrix of right vectors
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagBisect(
   ENUM_SVDBIDIAG_Z     jobz,         // how to compute left vectors
   ENUM_BLAS_RANGE      range,        // subset of computed singular values
   double               lower,        // lower limit of the subset
   double               upper,        // upper limit of the subset
   vectorf&             S,            // vector of computed singular values
   matrixf&             U,            // U matrix of computed left vectors
   matrixf&             VT            // VT transposed matrix of right vectors
   );
```

Parameters

jobz

[in] ENUM_SVDBIDIAG_Z enumeration value that determines how the left singular vectors should be computed.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable singular values and vectors.

lower

[in] The lower limit of singular values subset; specified depending on the value of the range parameter.

upper

[in] The upper limit of singular values subset; specified depending on the value of the range parameter.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

V

[out] Transposed matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

When BLASRANGE_A is set, all singular values are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those singular values (and their vectors) that fall within the range of real values specified by the 'lower' and 'upper' parameters are computed.

With the BLASRANGE_I value, only those singular values (and their vectors) that fall within the range of integer indices specified by the 'lower' and 'upper' parameters are computed. For example, with lower=0 and upper=2, only the first three singular values are computed.

A bidiagonal matrix is a square matrix with non-zero main diagonal and one of the sub-diagonals.

Upper bidiagonal matrix

```
[[x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x],
 [0, 0, 0, 0, x]]
```

Lower bidiagonal matrix

```
[[x, 0, 0, 0, 0],
 [x, x, 0, 0, 0],
 [0, x, x, 0, 0],
 [0, 0, x, x, 0],
 [0, 0, 0, x, x]]
```

ENUM_SVDBIDIAG_Z

An enumeration defining how left singular vectors should be computed.

ID	Description
SVDJOBZ_V	Compute singular values and singular vectors.
SVDJOBZ_N	Compute singular values only.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

See also

SingularValueDecompositionDC, SingularValueDecompositionQR

SingularValueDecompositionBidiagQR

Computes the singular value decomposition of a general matrix that has been reduced to bidiagonal form by the method ReduceToBidiagonal. Lapack function BDSQR.

Computing for type matrix<double>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,         // orthogonal matrix Q
   matrix&          PT,        // transposed matrix P
   vector&          S,         // vector of computed singular values
   matrix&          U,         // matrix U of computed left vectors
   matrix&          VT         // transposed matrix V of right vectors
   );
```

Computing for type matrix<float>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,         // orthogonal matrix Q
   matrix&          PT,        // transposed matrix P
   vectorf&         S,         // vector of computed singular values
   matrixf&         U,         // matrix U of computed left vectors
   matrixf&         VT         // transposed matrix V of right vectors
   );
```

Computing for type matrix<complex>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrix&          Q,         // orthogonal matrix Q
   matrix&          PH,        // Hermitian conjugated matrix P
   vector           S,         // vector of computed singular values
   matrixc          U,         // matrix U of computed left vectors
   matrixc          VH         // Hermitian conjugated matrix V of right vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrix::SingularValueDecompositionBidiagQR(
   matrixf&         Q,         // orthogonal matrix Q
   matrixf&         PH,        // Hermitian conjugated matrix P
   vectorf&         S,         // vector of computed singular values
   matrixcf&        U,         // matrix U of computed left vectors
   matrixcf&        VH         // Hermitian conjugated matrix V of right vectors
   );
```

Parameters

Q

[in] Orthogonal matrix Q produced by method ReflectBidiagonalToQP. If matrix Q has zero size, then left vectors U are not calculated.

PT

[in] Transposed (or hermitian conjugated) matrix P produced by method ReflectBidiagonalToQP. If matrix PT has zero size, then right vectors VT are not calculated.

S

[out] Vector of singular values.

U

[out] Matrix of left singular vectors.

VT

[out] Matrix of right singular vectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

BDSQR computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm. The SVD of B has the form

B = Q * S * P**T

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P**T*VT instead of P**T, for given real input matrices U and VT. When U and VT are the orthogonal matrices that reduce a general matrix A to bidiagonal form: A = U*B*VT, as computed by GEBRD, then

A = (U*Q) * S * (P**T*VT)

is the SVD of A.

Eigen Values

The section features functions for computing eigenvalues and eigenvectors. It describes methods for solving standard linear algebra problems using the LAPACK library algorithms. These functions are efficient for matrix analysis, diagonalization, system stabilization, and other tasks.

- **EigenSolver**: The function is designed to compute the eigenvalues and eigenvectors of an arbitrary square matrix using the classical algorithm represented by the GEEV lapack function. This method is applied to a wide range of matrices, allowing the decomposition of matrices into their eigenvalues and eigenvectors.
- **EigenSymmetricDC**: The function for computing eigenvalues and eigenvectors of symmetric or Hermitian matrices using the divide-and-conquer algorithm. The lapack functions SYEVD and HEEVD enable the efficient handling of symmetric or Hermitian matrices, providing faster and more accurate processing of such matrices.

Function	Action
EigenSolver	Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV).
EigenSolverX	Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX).
EigenSolverShur	Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition.
EigenSolver2	Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV).
EigenSolver2X	Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size.
EigenSolver2Shur	Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES).
EigenSolver2Blocked	Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2.

Function	Action
EigenSolver2ShurBlocked	Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3).
EigenSymmetricDC	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD).
EigenSymmetricQR	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV).
EigenSymmetricRobust	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR).
EigenSymmetricBisect	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX).
EigenSymmetricDC2s	Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm (lapack functions SYEVD_2STAGE, HEEVD_2STAGE).
EigenSymmetricQR2s	Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal (lapack functions SYEV_2STAGE, HEEV_2STAGE).
EigenSymmetricRobust2s	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR_2STAGE, HEEVR_2STAGE).
EigenSymmetricBisect2s	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the bisection algorithm (lapack functions SYEVX_2STAGE, HEEVX_2STAGE).

Function	Action
EigenSymmetric2DC	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.
EigenSymmetric2QR	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.
EigenSymmetric2Bisect	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.
EigenTridiagonalDC	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEVD).
EigenTridiagonalQR	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEV).
EigenTridiagonalRobust	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack function STEVR).
EigenTridiagonalBisect	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the bisection algorithm (lapack function STEVX).
EigenTridiagonalQL	Compute all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm (lapack function STERF).
EigenTridiagonalDCQ	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEDC).
EigenTridiagonalQRQ	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEQR).
EigenTridiagonalPosDefQ	Compute eigenvalues and eigenvectors of a symmetric positive definite (положительно определённая) tridiagonal matrix using the QR algorithm (lapack function PTEQR).

General Matrices

Functions for calculating eigenvalues and eigenvectors of a square matrix using classical algorithms. It provides various methods for working with both real and complex matrices, allowing you to solve linear algebra problems with a choice of methods for calculating eigenvectors.

Function	Action
EigenSolver	Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV).
EigenSolverX	Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX).
EigenSolverShur	Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition.
EigenSolver2	Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV).
EigenSolver2X	Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size.
EigenSolver2Shur	Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES).
EigenSolver2Blocked	Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2.
EigenSolver2ShurBlocked	Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3).

EigenSolver

Compute eigenvalues and eigenvectors of a regular square matrix using the classical algorithm (lapack function GEEV).

Computing for type matrix<double>

```
bool  matrix::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                    // compute left and right eigenvecto
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrix&               left_eigenvectors,       // matrix of computed left vectors
   matrix&               right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                    // compute left and right eigenvecto
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixf&              left_eigenvectors,       // matrix of computed left vectors
   matrixf&              right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                    // compute left and right vectors
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrixc&              left_eigenvectors,       // matrix of computed left vectors
   matrixc&              right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver(
   ENUM_EIG_VECTORS      jobv,                    // compute left and right vectors
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixcf&             left_eigenvectors,       // matrix of computed left vectors
   matrixcf&             right_eigenvectors       // matrix of computed right vectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

EV

[out] Vector of eigenvalues.

left_eigenvectors

[out] Matrix of left eigenvectors.

right_eigenvectors

[out] Matrix of right eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

If EIGVECTORS_N is set, the left and right vectors are not computed. Only eigenvalues are computed.

With EIGVECTORS_L, only left eigenvectors are computed, right eigenvectors are not computed.

When EIGVECTORS_R is set, only the right eigenvectors are computed, the left vectors are not computed.

With EIGVECTORS_LR, the left and right eigenvectors are computed, Eigenvalues are always computed.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

ENUM_EIG_VECTORS

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are calculated, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

EigenSolverX

Compute eigenvalues and eigenvectors of a regular square matrix in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GEEVX).

Computing for type matrix<double>

```
bool  matrix::EigenSolverX(
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrix&               left_eigenvectors,       // matrix of computed left vectors
   matrix&               right_eigenvectors       // matrix of computed right vectors
   matrix&               shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vector&               scale,                   // details of permutations and scali
   double&               ab_norm,                 // 1-norm of balanced matrix
   vector&               rconde,                  // vector of reciprocal condition nu
   vector&               rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolverX(
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixf&              left_eigenvectors,       // matrix of computed left vectors
   matrixf&              right_eigenvectors       // matrix of computed right vectors
   matrixf&              shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vectorf&              scale,                   // details of permutations and scali
   float&                ab_norm,                 // 1-norm of balanced matrix
   vectorf&              rconde,                  // vector of reciprocal condition nu
   vectorf&              rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolverX(
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrixc&              left_eigenvectors,       // matrix of computed left vectors
   matrixc&              right_eigenvectors       // matrix of computed right vectors
   matrixc&              shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vector&               scale,                   // details of permutations and scal
   double&               ab_norm,                 // 1-norm of balanced matrix
   vector&               rconde,                  // vector of reciprocal condition nu
   vector&               rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolverX(
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixcf&             left_eigenvectors,       // matrix of computed left vectors
   matrixcf&             right_eigenvectors       // matrix of computed right vectors
   matrixcf&             shur_matrix,             // balanced matrix in Schur form
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vectorf&              scale,                   // details of permutations and scal
   float&                ab_norm,                 // 1-norm of balanced matrix
   vectorf&              rconde,                  // vector of reciprocal condition nu
   vectorf&              rcondv                   // vector of reciprocal condition nu
   );
```

Parameters

balance

[in] Value from the ENUM_EIG_BALANCE enumeration which determines the need and method for balancing the input matrix; it is used to improve the conditioning of the eigenvalues and eigenvectors.

jobv

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

sense

[in] Value from the ENUM_EIG_SENSE enumeration determining the need to compute reciprocal condition numbers.

eigen_values

[out] Vector of eigenvalues.

left_eigenvectors

[out] Matrix of left eigenvectors.

right_eigenvectors

[out] Matrix of right eigenvectors.

shur_matrix

[out] Balanced matrix in Schur form; the matrix is not filled if neither left nor right eigenvectors are computed.

ilo

[out] Subscript of the balanced matrix; the matrix is not filled if no balancing is applied.

ihi

[out] Superscript of the balanced matrix; the matrix is not filled if no balancing is applied.

scale

[out] Vector of details of permutations and scaling when balancing the input matrix.

Details of the permutations and scaling factors applied when balancing A.

If P(j) is the index of the row and column interchanged with row and column j, and D(j) is the scaling factor applied to row and column j, then

scale(j) = P(j), for j = 1,...,ilo-1

= D(j), for j = ilo,...,ihi

= P(j) for j = ihi+1,..., n.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

ab_norm

[out] 1-norm of the balanced matrix (the maximum of the sum of absolute values of elements in any of the matrix columns).

rconde

[out] Vector of reciprocal condition numbers for each eigenvalue; it is computed if the 'sense' parameter is set to 'E' or 'B'.

rcondv

[out] Vector of reciprocal condition numbers for each eigenvector; it is computed if the 'sense' parameter is set to 'V' or 'B'.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Input matrix balancing depends on the value of the '*balance*' parameter.

ENUM_EIG_BALANCE

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGBALANCE_N	Do not diagonally scale or permute
EIGBALANCE_P	Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale
EIGBALANCE_S	Diagonally scale the matrix. Do not permute
EIGBALANCE_B	Both diagonally scale and permute

ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

ENUM_EIG_SENSE

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGSENSE_N	None of reciprocal condition numbers are computed
EIGSENSE_E	Computed for eigenvalues only
EIGSENSE_V	Computed for right eigenvectors only
EIGSENSE_B	Computed for eigenvalues and right eigenvectors

EigenSolverShur

Compute eigenvalues, upper triangular matrix in Schur form, and matrix of Schur vectors (lapack function GEES). See also Schur decomposition.

Computing for type matrix<double>

```
bool  matrix::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                   // method to compute Shur vectors
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrix&               shur_matrix,             // matrix in Schur form
   matrix&               shur_vectors             // matrix of Schur vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                   // method to compute Shur vectors
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixf&              shur_matrix,             // matrix in Schur form
   matrixf&              shur_vectors             // matrix of Schur vectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                   // method to compute Shur vectors
   vectorc&              eigen_values,            // vector of computed eigenvalues
   matrixc&              shur_matrix,             // matrix in Schur form
   matrixc&              shur_vectors             // matrix of Schur vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolverShur(
   ENUM_EIG_VECTORS      jobvs,                   // method to compute Shur vectors
   vectorcf&             eigen_values,            // vector of computed eigenvalues
   matrixcf&             shur_matrix,             // matrix in Schur form
   matrixcf&             shur_vectors             // matrix of Schur vectors
   );
```

Parameters

jobvs

[in] Value from the ENUM_EIG_SHUR enumeration, which defines the method for computing Shur vectors.

eigen_values

[out] Vector of eigenvalues.

shur_matrix

[out] Upper triangular Schur matrix (Schur form for the input matrix).

shur_vectors

[out] Matrix of Schur vectors; it is not computed if the *jobvs* parameter is set to N.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the *jobvs* parameter values.

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

EigenSolverShur

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGSHUR_N	Schur vectors are not computed
EIGSHUR_V	Schur vectors are computed

EigenSolver2

Compute generalized eigenvalues and eigenvectors for a pair of ordinary square matrices (lapack function GGEV). Both matrices must be the same size.

Computing for type matrix<double>

```
bool  matrix::EigenSolver2(
   matrix&                 B,                        // second matrix in the pair
   ENUM_EIG_VECTORS        jobv,                     // method to compute right and left
   vectorc&                alpha,                    // vector of computed eigenvalues
   vector&                 beta,                     // vector of eigenvalue divisors
   matrix&                 left_eigenvectors,        // matrix of computed left vectors
   matrix&                 right_eigenvectors        // matrix of computed right vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolver2(
   matrix&                 B,                        // second matrix in the pair
   ENUM_EIG_VECTORS        jobv,                     // method to compute right and left
   vectorcf&               alpha,                    // vector of computed eigenvalues
   vectorf&                beta,                     // vector of eigenvalue divisors
   matrixf&                left_eigenvectors,        // matrix of computed left vectors
   matrixf&                right_eigenvectors        // matrix of computed right vectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolver2(
   matrixc&                B,                        // second matrix in the pair
   ENUM_EIG_VECTORS        jobv,                     // method to compute right and left
   vectorc&                alpha,                    // vector of computed eigenvalues
   vectorc&                beta,                     // vector of eigenvalue divisors
   matrixc&                left_eigenvectors,        // matrix of computed left vectors
   matrixc&                right_eigenvectors        // matrix of computed right vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2(
   matrixcf&               B,                        // second matrix in the pair
   ENUM_EIG_VECTORS        jobv,                     // method to compute right and left
   vectorcf&               alpha,                    // vector of computed eigenvalues
   vectorcf&               beta,                     // vector of eigenvalue divisors
   matrixcf&               left_eigenvectors,        // matrix of computed left vectors
   matrixcf&               right_eigenvectors        // matrix of computed right vectors
   );
```

Parameters

B

[in] The second matrix in the pair.

jobv

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

alpha

[out] Vector of eigenvalues.

beta

[out] Vector of eigen value divisors.

left_eigenvectors

[out] Matrix of left eigenvectors.

right_eigenvectors

[out] Matrix of right eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

A * v(j) = lambda(j) * B * v(j).

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies

u(j)**H * A = lambda(j) * u(j)**H * B .

where u(j)**H is the conjugate-transpose of u(j).

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

EigenSolver2X

Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices in Expert mode, i.e. with the ability to influence the computation algorithm and the ability to obtain accompanying computation data (lapack function GGEVX). Both matrices must be the same size.

Optionally, it also computes a balancing transformation to improve the conditioning of the eigenvalues and eigenvectors (ILO, IHI, LSCALE, RSCALE, ABNRM, and BBNRM), reciprocal condition numbers for the eigenvalues (RCONDE), and reciprocal condition numbers for the right eigenvectors (RCONDV).

Computing for type matrix<double>

```
bool  matrix::EigenSolver2X(
   matrix&               B,                       // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorc&              alpha,                   // vector of computed eigenvalues
   vector&               beta,                    // vector of eigenvalue divisors
   matrix&               left_eigenvectors,       // matrix of computed left vectors
   matrix&               right_eigenvectors       // matrix of computed right vectors
   matrix&               shur_matrix1,            // the first part of the real Schur
   matrix&               shur_matrix2,            // the second part of the real Schu
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vector&               lscale,                  // details of the permutations and
   vector&               rscale,                  // details of the permutations and
   double&               ab_norm,                 // one-norm of balanced input matri
   double&               bb_norm,                 // one-norm of balanced second matri
   vector&               rconde,                  // vector of reciprocal condition nu
   vector&               rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolver2X(
   matrixf&              B,                       // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorcf&             alpha,                   // vector of computed eigenvalues
   vectorf&              beta,                    // vector of eigenvalue divisors
   matrixf&              left_eigenvectors,       // matrix of computed left vectors
   matrixf&              right_eigenvectors       // matrix of computed right vectors
   matrixf&              shur_matrix1,            // the first part of the real Schur
   matrixf&              shur_matrix2,            // the second part of the real Schu
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vectorf&              lscale,                  // details of the permutations and
   vectorf&              rscale,                  // details of the permutations and s
   float&                ab_norm,                 // one-norm of balanced input matrix
   float&                bb_norm,                 // one-norm of balanced second matri
   vectorf&              rconde,                  // vector of reciprocal condition nu
   vectorf&              rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSolver2X(
   matrixc&              B,                       // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorc&              alpha,                   // vector of computed eigenvalues
   vectorc&              beta,                    // vector of eigenvalue divisors
   matrixc&              left_eigenvectors,       // matrix of computed left vectors
   matrixc&              right_eigenvectors       // matrix of computed right vectors
   matrixc&              shur_matrix1,            // the first part of the real Schur
   matrixc&              shur_matrix2,            // the second part of the real Schur
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vector&               lscale,                  // details of the permutations and s
   vector&               rscale,                  // details of the permutations and s
   double&               ab_norm,                 // one-norm of balanced input matrix
   double&               bb_norm,                 // one-norm of balanced second matri
   vector&               rconde,                  // vector of reciprocal condition nu
   vector&               rcondv                   // vector of reciprocal condition nu
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2X(
   matrixcf&             B,                       // second matrix in the pair
   ENUM_EIG_BALANCE      balance,                 // input matrix balancing method
   ENUM_EIG_VECTORS      jobv,                    // determines computation of right a
   ENUM_EIG_SENSE        sense,                   // determines computation of recipro
   vectorcf&             alpha,                   // vector of computed eigenvalues
   vectorf&              beta,                    // vector of eigenvalue divisors
   matrixcf&             left_eigenvectors,       // matrix of computed left vectors
   matrixcf&             right_eigenvectors       // matrix of computed right vectors
   matrixcf&             shur_matrix1,            // the first part of the real Schur
   matrixcf&             shur_matrix2,            // the second part of the real Schur
   long&                 ilo,                     // subscript of balanced matrix
   long&                 ihi,                     // superscript of balanced matrix
   vectorf&              lscale,                  // details of the permutations and s
   vectorf&              rscale,                  // details of the permutations and s
   float&                ab_norm,                 // one-norm of balanced input matrix
   float&                 bb_norm,               // one-norm of balanced second matri
   vectorf&               rconde,                // vector of reciprocal condition nu
   vectorf&               rcondv                 // vector of reciprocal condition nu
   );
```

Parameters

B

[in] The second matrix in the pair.

balance

[in] Value from the ENUM_EIG_BALANCE enumeration which determines the need and method for balancing the input matrix; it is used to improve the conditioning of the eigenvalues and eigenvectors.

jobv

[in] Value from the ENUM_EIG_VECTORS enumeration which determines the method for computing left and right eigenvectors.

sense

[in] Value from the ENUM_EIG_SENSE enumeration which determines the need to compute reciprocal condition numbers.

eigen_values

[out] Vector of eigenvalues.

left_eigenvectors

[out] Matrix of left eigenvectors.

right_eigenvectors

[out] Matrix of right eigenvectors.

shur_matrix1, shur_matrix2

[out] 2 parts of balanced matrix in Schur form; the matrix is not filled if neither left nor right eigenvectors are computed.

ilo

[out] Subscript of the balanced matrix; the matrix is not filled if no balancing is applied.

ihi

[out] Superscript of the balanced matrix; the matrix is not filled if no balancing is applied.

lscale

[out] Vector contains details of the permutations and scaling factors applied to the left side of A and B.

If PL(j) is the index of the row interchanged with row j, and DL(j) is the scaling factor applied to row j, then

lscale(j) = PL(j), for j = 1,..., ilo-1

= DL(j), for j = ilo,...,ihi

= PL(j) for j = ihi+1,..., n.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

rscale

[out] Vector contains details of the permutations and scaling factors applied to the right side of A and B.

If PR(j) is the index of the column interchanged with column j, and DR(j) is the scaling factor applied to column j, then

rscale(j) = PR(j), for j = 1,..., ilo-1

= DR(j), for j = ilo,...,ihi

= PR(j) for j = ihi+1,..., n.

The order in which the interchanges are made is n to ihi+1, then 1 to ilo-1.

ab_norm

[out] One-norm of the balanced input matrix (the maximum of the sum of absolute values of elements in any of the matrix columns).

bb_norm

[out] One-norm of balanced second matrix B.

rconde

[out] Vector of reciprocal condition numbers for each eigenvalue; it is computed if the 'sense' parameter is set to 'E' or 'B'.

rcondv

[out] Vector of reciprocal condition numbers for each eigenvector; it is computed if the 'sense' parameter is set to 'V' or 'B'.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

Input matrices balancing depends on the value of the *balance* parameter.

ENUM_EIG_BALANCE

An enumeration that specifies whether the matrices should be balanced.

ID	Description
EIGBALANCE_N	Do not diagonally scale or permute.
EIGBALANCE_P	Perform permutations to make the matrix more nearly upper triangular. Do not diagonally scale.
EIGBALANCE_S	Diagonally scale the matrix. Do not permute.
EIGBALANCE_B	Both diagonally scale and permute.

ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

ENUM_EIG_SENSE

An enumeration determining the need to compute reciprocal condition numbers.

ID	Description
EIGSENSE_N	None of reciprocal condition numbers are computed.
EIGSENSE_E	Computed for eigenvalues only.
EIGSENSE_V	Computed for right eigenvectors only.
EIGSENSE_B	Computed for eigenvalues and right eigenvectors.

EigenSolver2Shur

Compute a pair of ordinary square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES).

Computes the generalized eigenvalues, the generalized real/complex Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (vsl and vsr) for a pair of n-by-n real/complex nonsymmetric matrices (A,B). This gives the generalized Schur factorization:

(A,B) = (vsl*S *vsrH, vsl*T*vsrH)

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T. The leading columns of vsl and vsr then form an orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

Computing for type matrix<double>

```
bool  matrix::EigenSolver2Shur(
   matrix&                 B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vector&                 alpha,     // vector of computed eigenvalues
   vector&                 beta,      // vector of eigenvalue divisors
   matrix&                 shur_s,    // matrix S in Schur form
   matrix&                 shur_t,    // matrix T in Schur form
   matrix&                 vsl,       // matrix of left Schur vectors vsl
   matrix&                 vsr        // matrix of right Schur vectors vsr
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolver2Shur(
   matrixf&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorcf&               alpha,     // vector of computed eigenvalues
   vectorf&                beta,      // vector of eigenvalue divisors
   matrixf&                shur_s,    // matrix S in Schur form
   matrixf&                shur_t,    // matrix T in Schur form
   matrixf&                vsl,       // matrix of left Schur vectors vsl
   matrixf&                vsr        // matrix of right Schur vectors vsr
   );
```

Computing for type matrix<complex>

```
bool  matrix::EigenSolver2Shur(
   matrixc&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS        jobvs,     // method to compute left and right vectors
   vectorc&                alpha,     // vector of computed eigenvalues
   vectorc&                beta,      // vector of eigenvalue divisors
   matrixc&                shur_s,    // matrix S in Schur form
   matrixc&                shur_t,    // matrix T in Schur form
   matrixc&                 vsl,      // matrix of left Schur vectors vsl
   matrixc&                 vsr       // matrix of right Schur vectors vsr
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2Shur(
   matrixcf&                B,        // second matrix in the pair
   ENUM_EIG_VECTORS         jobvs,    // method to compute left and right vectors
   vectorcf&                alpha,    // vector of computed eigenvalues
   vectorcf&                beta,     // vector of eigenvalue divisors
   matrixcf&                shur_s,   // matrix S in Schur form
   matrixcf&                shur_t,   // matrix T in Schur form
   matrixcf&                vsl,      // matrix of left Schur vectors vsl
   matrixcf&                vsr       // matrix of right Schur vectors vsr
   );
```

Parameters

B

[out] The second matrix in the pair.

jobvs

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

alpha

[out] Vector of eigenvalues.

beta

[out] Vector of eigenvalue divisors.

shur_s

[out] Matrix S, block upper triangular Schur matrix (Schur form for the input matrix).

shur_t

[out] Matrix T, block upper triangular Schur matrix (Schur form for the second matrix in the pair).

vsl

[out] Matrix of left Schur vectors.

vsr

[out] Matrix of right Schur vectors.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

Computation depends on the *jobvs* parameter values.

The second matrix in the pair must be the same size as the first (input) one.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

EigenSolverShur

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

EigenSolver2Blocked

Compute generalized eigenvalues and eigenvectors for a pair of regular square matrices using a block algorithm (lapack function GGEV3). Both matrices must be the same size. The method parameters are exactly the same as EigenSolver2.

Computing for type matrix<double>

```
bool  matrix::EigenSolver2Blocked(
   matrix&               B,                       // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                    // method to compute right and left
   vectorc&              alpha,                   // vector of computed eigenvalues
   vector&               beta,                    // vector of eigenvalue divisors
   matrix&               left_eigenvectors,       // matrix of computed left vectors
   matrix&               right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSolver2Blocked(
   matrix&               B,                       // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                    // method to compute right and left
   vectorcf&             alpha,                   // vector of computed eigenvalues
   vectorf&              beta,                    // vector of eigenvalue divisors
   matrixf&              left_eigenvectors,       // matrix of computed left vectors
   matrixf&              right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<complex>

```
bool  matrix::EigenSolver2Blocked(
   matrixc&              B,                       // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                    // method to compute right and left
   vectorc&              alpha,                   // vector of computed eigenvalues
   vectorc&              beta,                    // vector of eigenvalue divisors
   matrixc&              left_eigenvectors,       // matrix of computed left vectors
   matrixc&              right_eigenvectors       // matrix of computed right vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSolver2(
   matrixcf&             B,                       // second matrix in the pair
   ENUM_EIG_VECTORS      jobv,                    // method to compute right and left
   vectorcf&             alpha,                   // vector of computed eigenvalues
   vectorcf&             beta,                    // vector of eigenvalue divisors
   matrixcf&             left_eigenvectors,       // matrix of computed left vectors
   matrixcf&             right_eigenvectors       // matrix of computed right vectors
   );
```

Parameters

B

[out] The second matrix in the pair.

jobv

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

alpha

[out] Vector of eigenvalues.

beta

[out] Vector of eigenvalue divisors.

left_eigenvectors

[out] Matrix of left eigenvectors.

righeft_eigenvectors

[out] Matrix of right eigenvectors.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

Computation depends on the value of the jobv parameter.

A generalized eigenvalue for a pair of matrices (A,B) is a scalar lambda or a ratio alpha/beta = lambda, such that A - lambda*B is singular. It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

The right eigenvector v(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies:

A * v(j) = lambda(j) * B * v(j).

The left eigenvector u(j) corresponding to the eigenvalue lambda(j) of (A,B) satisfies:

u(j)**H * A = lambda(j) * u(j)**H * B .

where u(j)**H is the conjugate-transpose of u(j).

Real (non-complex) matrices can have a complex solution. Therefore, the input vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

ENUM_EIG_VECTORS

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.

ID	Description
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

EigenSolver2ShurBlocked

Compute a pair of regular square matrices of generalized eigenvalues, generalized eigenvectors, generalized Schur forms, as well as left and right Schur vectors (lapack function GGES3).

Computes the generalized eigenvalues, the generalized real/complex Schur form (S,T), optionally, the left and/or right matrices of Schur vectors (VSL and VSR) for a pair of n-by-n real/complex nonsymmetric matrices (A,B). This gives the generalized Schur factorization:

(A,B) = (vsl*S *vsrH, vsl*T*vsrH)

Optionally, it also orders the eigenvalues so that a selected cluster of eigenvalues appears in the leading diagonal blocks of the upper quasi-triangular matrix S and the upper triangular matrix T. The leading columns of vsl and vsr then form an orthonormal/unitary basis for the corresponding left and right eigenspaces (deflating subspaces).

Computing for type matrix<double>

```
bool  matrix::EigenSolver2ShurBlocked(
   matrix&                  B,         // second matrix in the pair
   ENUM_EIG_VECTORS         jobvs,     // method to compute left and right vectors
   vectorc&                 alpha,     // vector of computed eigenvalues
   vector&                  beta,      // vector of eigenvalue divisors
   matrix&                  shur_s,    // matrix S in Schur form
   matrix&                  shur_t,    // matrix T in Schur form
   matrix&                  vsl,       // matrix of left Schur vectors VSL
   matrix&                  vsr        // matrix of right Schur vectors VSR
   );
```

Computing for type matrix<float>

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixf&                 B,         // second matrix in the pair
   ENUM_EIG_VECTORS         jobvs,     // method to compute left and right vectors
   vectorcf&                alpha,     // vector of computed eigenvalues
   vectorf&                 beta,      // vector of eigenvalue divisors
   matrixf&                 shur_s,    // matrix S in Schur form
   matrixf&                 shur_t,    // matrix T in Schur form
   matrixf&                 vsl,       // matrix of left Schur vectors VSL
   matrixf&                 vsr        // matrix of right Schur vectors VSR
   );
```

Computing for type matrix<complex>

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixc&                 B,         // second matrix in the pair
   ENUM_EIG_VECTORS         jobvs,     // method to compute left and right vectors
   vectorc&                 alpha,     // vector of computed eigenvalues
   vectorc&                 beta,      // vector of eigenvalue divisors
   matrixc&                 shur_s,    // matrix S in Schur form
   matrixc&                 shur_t,    // matrix T in Schur form
   matrixc&                 vsl,       // matrix of left Schur vectors VSL
   matrixc&                 vsr        // matrix of right Schur vectors VSR
   );
```

Computing for type matrix<complexf>

```
bool  matrix::EigenSolver2ShurBlocked(
   matrixcf&                B,         // second matrix in the pair
   ENUM_EIG_VECTORS         jobvs,     // method to compute left and right vectors
   vectorcf&                alpha,     // vector of computed eigenvalues
   vectorcf&                beta,      // vector of eigenvalue divisors
   matrixcf&                shur_s,    // matrix S in Schur form
   matrixcf&                shur_t,    // matrix T in Schur form
   matrixcf&                vsl,       // matrix of left Schur vectors VSL
   matrixcf&                vsr        // matrix of right Schur vectors VSR
   );
```

Parameters

B

[in] The second matrix in the pair.

jobvs

[in] ENUM_EIG_VECTORS enumeration value which determines the method for computing left and right eigenvectors.

alpha

[out] Vector of eigenvalues.

beta

[out] Vector of eigenvalue divisors.

shur_s

[out] Matrix S, block upper triangular Schur matrix (Schur form for the input matrix).

shur_t

[out] Matrix T, block upper triangular Schur matrix (Schur form for the second matrix in the pair).

vsl

[out] Matrix of left Schur vectors VSL.

vsr

[out] Matrix of right Schur vectors VSR.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

Computation depends on the *jobvs* parameter values.

The second matrix in the pair must be the same size as the first (input) one.

Real (non-complex) matrices can have a complex solution. Therefore, the vector of eigenvalues must be complex. In case of a complex solution, the error code is set to 4019 (ERR_MATH_OVERFLOW). Otherwise, only the real parts of the complex values of the eigenvalue vector should be used.

EigenSolverShur

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVECTORS_N	Only eigenvalues are computed, without vectors.
EIGVECTORS_L	Only left eigenvectors are computed.
EIGVECTORS_R	Only right eigenvectors are computed.
EIGVECTORS_LR	Left and right eigenvectors are computed, eigenvalues are always computed.

Symmetric Matrices

Functions for computing eigenvalues and eigenvectors of symmetric or Hermitian matrices using the divide and conquer algorithm, making the process efficient and fast. These methods can be applied to matrices of different data types, including real and complex numbers.

Function	Action
EigenSymmetricDC	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD).
EigenSymmetricQR	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the QR algorithm (lapack functions SYEV, HEEV).
EigenSymmetricRobust	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR).
EigenSymmetricBisect	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugate) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX).
EigenSymmetricDC2s	Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm (lapack functions SYEVD_2STAGE, HEEVD_2STAGE).
EigenSymmetricQR2s	Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal (lapack functions SYEV_2STAGE, HEEV_2STAGE).
EigenSymmetricRobust2s	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR_2STAGE, HEEVR_2STAGE).
EigenSymmetricBisect2s	Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction

Function	Action
	to tridiagonal then using the bisection algorithm (lapack functions SYEVX_2STAGE, HEEVX_2STAGE).
EigenSymmetric2DC	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.
EigenSymmetric2QR	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.
EigenSymmetric2Bisect	Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

EigenSymmetricDC

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the divide-and-conquer algorithm (lapack functions SYEVD, HEEVD).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vector&                eigen_values,       // vector of computed eigenvalues
   matrix&                eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixf&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vector&                eigen_values,       // vector of computed eigenvalues
   matrixc&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixcf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenSymmetricQR

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the QR algorithm (lapack functions SYEV, HEEV).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrix&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixf&              eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vector&               eigen_values,      // vector of computed eigenvalues
   matrixc&              eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricQR(
   ENUM_EIG_VALUES       jobv,              // compute eigenvectors or not
   vectorf&              eigen_values,      // vector of computed eigenvalues
   matrixcf&             eigen_vectors      // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

EigenSymmetricRobust

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR, HEEVR).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   double                lower,              // lower bound of the subset
   double                upper,              // Upper bound of the subset
   double                abstol,             // absolute error tolerance
   vector&               eigen_values,       // vector of computed eigenvectors
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   float                 lower,              // lower bound of the subset
   float                 upper,              // upper bound of the subset
   float                 abstol,             // absolute error tolerance
   vectorf&              eigen_values,       // vector of computed eigenvectors
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   double                lower,              // lower bound of the subset
   double                upper,              // Upper bound of the subset
   double                abstol,             // absolute error tolerance
   vector&               eigen_values,       // vector of computed eigenvectors
   matrixc&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricRobust(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   float                 lower,              // lower bound of the subset
   float                 upper,              // Upper bound of the subset
   float                 abstol,             // absolute error tolerance
   vectorf&              eigen_values,        // vector of computed eigenvectors
   matrixcf&             eigen_vectors        // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenSymmetricBisect

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the bisection algorithm (lapack functions SYEVX, HEEVX).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricBisect(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   double                 lower,             // lower bound of the subset
   double                 upper,             // Upper bound of the subset
   double                 abstol,            // absolute error tolerance
   vector&                eigen_values,      // vector of computed eigenvectors
   matrix&                eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricBisect(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   float                  lower,             // lower bound of the subset
   float                  upper,             // upper bound of the subset
   float                  abstol,            // absolute error tolerance
   vectorf&               eigen_values,      // vector of computed eigenvectors
   matrixf&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricBisect(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   double                 lower,             // lower bound of the subset
   double                 upper,             // Upper bound of the subset
   double                 abstol,            // absolute error tolerance
   vector&                eigen_values,      // vector of computed eigenvectors
   matrixc&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricBisect(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   float                  lower,             // lower bound of the subset
   float                  upper,             // Upper bound of the subset
   float                  abstol,            // absolute error tolerance
   vectorf&               eigen_values,      // vector of computed eigenvectors
   matrixcf&                eigen_vectors        // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

V

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenSymmetricDC2s

Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal. If eigenvectors are desired, it uses a divide and conquer algorithm (lapack functions SYEVD_2STAGE, HEEVD_2STAGE).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricDC2s(
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated. Not available in this release.
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenSymmetricQR2s

Compute all eigenvalues and, optionally, eigenvectors of a real symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal (lapack functions SYEV_2STAGE, HEEV_2STAGE).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,                 // compute eigenvectors or not
   vector&               eigen_values,         // vector of computed eigenvalues
   matrix&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,                 // compute eigenvectors or not
   vectorf&              eigen_values,         // vector of computed eigenvalues
   matrixf&              eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,                 // compute eigenvectors or not
   vector&               eigen_values,         // vector of computed eigenvalues
   matrixc&              eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricQR2s(
   ENUM_EIG_VALUES       jobv,                 // compute eigenvectors or not
   vectorf&              eigen_values,         // vector of computed eigenvalues
   matrixcf&             eigen_vectors         // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated. Not available in this release.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

EigenSymmetricRobust2s

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the Multiple Relatively Robust Representations, MRRR algorithm (lapack functions SYEVR_2STAGE, HEEVR_2STAGE).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES      jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE      range,              // subset of eigenvalues to compute
   double               lower,              // lower bound of the subset
   double               upper,              // upper bound of the subset
   double               abstol,             // absolute error tolerance
   vector&              eigen_values,       // vector of computed eigenvectors
   matrix&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES      jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE      range,              // subset of eigenvalues to compute
   float                lower,              // lower bound of the subset
   float                upper,              // upper bound of the subset
   float                abstol,             // absolute error tolerance
   vectorf&             eigen_values,       // vector of computed eigenvectors
   matrixf&             eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES      jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE      range,              // subset of eigenvalues to compute
   double               lower,              // lower bound of the subset
   double               upper,              // upper bound of the subset
   double               abstol,             // absolute error tolerance
   vector&              eigen_values,       // vector of computed eigenvectors
   matrixc&             eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricRobust2s(
   ENUM_EIG_VALUES      jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE      range,              // subset of eigenvalues to compute
   float                lower,              // lower bound of the subset
   float                upper,              // upper bound of the subset
   float                abstol,             // absolute error tolerance
   vectorf&                eigen_values,        // vector of computed eigenvectors
   matrixcf&               eigen_vectors        // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated. Not available in this release.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenSymmetricBisect2s

Compute eigenvalues and eigenvectors of a symmetric or Hermitian (complex conjugated) matrix using the 2stage technique for the reduction to tridiagonal then using the bisection algorithm (lapack functions SYEVX_2STAGE, HEEVX_2STAGE).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   double                 lower,             // lower bound of the subset
   double                 upper,             // Upper bound of the subset
   double                 abstol,            // absolute error tolerance
   vector&                eigen_values,      // vector of computed eigenvectors
   matrix&                eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   float                  lower,             // lower bound of the subset
   float                  upper,             // upper bound of the subset
   float                  abstol,            // absolute error tolerance
   vectorf&               eigen_values,      // vector of computed eigenvectors
   matrixf&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   double                 lower,             // lower bound of the subset
   double                 upper,             // upper bound of the subset
   double                 abstol,            // absolute error tolerance
   vector&                eigen_values,      // vector of computed eigenvectors
   matrixc&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetricBisect2s(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,             // subset of eigenvalues to compute
   float                  lower,             // lower bound of the subset
   float                  upper,             // upper bound of the subset
   float                  abstol,            // absolute error tolerance
   vectorf&                 eigen_values,        // vector of computed eigenvectors
   matrixcf&                eigen_vectors        // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

V

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated. In the current OpenBLAS implementation, this value is not supported. Attempting to use it will result in error 4003 (ERR_INVALID_PARAMETER).

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated. Not available in this release.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenSymmetric2DC

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. If eigenvectors are desired, it uses a divide-and-conquer algorithm (lapack functions SYGVD, HEGVD).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,                 // the problem type to be solved
   ENUM_EIG_VALUES       jobv,                  // compute eigenvectors or not
   matrix&               B,                     // second matrix
   vector&               eigen_values,          // vector of computed eigenvalues
   matrix&               eigen_vectors,         // matrix of computed eigenvectors
   matrix&               triangular_factor      // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,                 // the problem type to be solved
   ENUM_EIG_VALUES       jobv,                  // compute eigenvectors or not
   matrixf&              B,                     // second matrix
   vectorf&              eigen_values,          // vector of computed eigenvalues
   matrixf&              eigen_vectors,         // matrix of computed eigenvectors
   matrixf&              triangular_factor      // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,                 // the problem type to be solved
   ENUM_EIG_VALUES       jobv,                  // compute eigenvectors or not
   matrixc&              B,                     // second matrix
   vector&               eigen_values,          // vector of computed eigenvalues
   matrixc&              eigen_vectors,         // matrix of computed eigenvectors
   matrixc&              triangular_factor      // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetric2DC(
   ENUM_EIGS2_TYPE       itype,                 // the problem type to be solved
   ENUM_EIG_VALUES       jobv,                  // compute eigenvectors or not
   matrixcf&             B,                     // second matrix
   vectorf&              eigen_values,          // vector of computed eigenvalues
   matrixcf&             eigen_vectors,         // matrix of computed eigenvectors
   matrixcf&             triangular_factor      // triangular factor from the Cholesky fa
   );
```

Parameters

itype

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

B

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

triangular_factor

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

ID	Description
EIGS2TYPE_1	1: A*x = (lambda)*B*x
EIGS2TYPE_2	2: A*B*x = (lambda)*x
EIGS2TYPE_3	3: B*A*x = (lambda)*x

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.

ID	Description
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenSymmetric2QR

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. This method uses QR algorithm (lapack functions SYGV, HEGV).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrix&               B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrix&               eigen_vectors,      // matrix of computed eigenvectors
   matrix&               triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixf&              B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixf&              eigen_vectors,      // matrix of computed eigenvectors
   matrixf&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixc&              B,                  // second matrix
   vector&               eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors,      // matrix of computed eigenvectors
   matrixc&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetric2QR(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   matrixcf&             B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors,      // matrix of computed eigenvectors
   matrixcf&             triangular_factor   // triangular factor from the Cholesky fa
   );
```

Parameters

itype

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

B

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

triangular_factor

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

ID	Description
EIGS2TYPE_1	1: A*x = (lambda)*B*x
EIGS2TYPE_2	2: A*B*x = (lambda)*x
EIGS2TYPE_3	3: B*A*x = (lambda)*x

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.

ID	Description
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenSymmetric2Bisect

Compute all the eigenvalues, and optionally, the eigenvectors of a generalized symmetric-definite eigenproblem, of the form A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

Here A and B are assumed to be symmetric (Hermitian) and B is also positive definite. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues. This method uses bisection algorithm (lapack functions SYGVX, HEGVX).

Computing for type matrix<double>

```
bool  matrix::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE        itype,              // the problem type to be solved
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,              // subset of eigenvalues to compute
   double                 lower,              // lower bound of the subset
   double                 upper,              // Upper bound of the subset
   double                 abstol,             // absolute error tolerance
   matrix&                B,                  // second matrix
   vector&                eigen_values,       // vector of computed eigenvalues
   matrix&                eigen_vectors,      // matrix of computed eigenvectors
   matrix&                triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE        itype,              // the problem type to be solved
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,              // subset of eigenvalues to compute
   float                  lower,              // lower bound of the subset
   float                  upper,              // upper bound of the subset
   float                  abstol,             // absolute error tolerance
   matrixf&               B,                  // second matrix
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixf&               eigen_vectors,      // matrix of computed eigenvectors
   matrixf&               triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE        itype,              // the problem type to be solved
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,              // subset of eigenvalues to compute
   double                 lower,              // lower bound of the subset
   double                 upper,              // Upper bound of the subset
   double                 abstol,             // absolute error tolerance
   matrixc&               B,                  // second matrix
   vector&                eigen_values,       // vector of computed eigenvalues
   matrixc&              eigen_vectors,      // matrix of computed eigenvectors
   matrixc&              triangular_factor   // triangular factor from the Cholesky fa
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenSymmetric2Bisect(
   ENUM_EIGS2_TYPE       itype,              // the problem type to be solved
   ENUM_EIG_VALUES       jobv,               // compute eigenvectors or not
   ENUM_BLAS_RANGE       range,              // subset of eigenvalues to compute
   float                 lower,              // lower bound of the subset
   float                 upper,              // Upper bound of the subset
   float                 abstol,             // absolute error tolerance
   matrixcf&             B,                  // second matrix
   vectorf&              eigen_values,       // vector of computed eigenvalues
   matrixcf&             eigen_vectors,      // matrix of computed eigenvectors
   matrixcf&             triangular_factor   // triangular factor from the Cholesky fa
   );
```

Parameters

itype

[in] ENUM_EIGS2_TYPE enumeration value which specified the problem type to be solved : A*x=(lambda)*B*x, A*Bx=(lambda)*x, or B*A*x=(lambda)*x.

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

B

[in] Second matrix B. Must be positive definite symmetric (or Hermitian conjugated) matrix.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

triangular_factor

[out] The triangular factor U or L from the Cholesky factorization B = U**T*U or B = L*L**T.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated). Second matrix B must be positive definite symmetric. If the input matrix and second matrix B are triangular, then both must be the same triangular, upper or lower.

ENUM_EIGS2_TYPE

An enumeration that specifies the problem type to be solved.

ID	Description
EIGS2TYPE_1	1: A*x = (lambda)*B*x
EIGS2TYPE_2	2: A*B*x = (lambda)*x
EIGS2TYPE_3	3: B*A*x = (lambda)*x

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

Tridiagonal Matrices

Functions for computing eigenvalues and eigenvectors of symmetric tridiagonal matrices using various algorithms. Each function implements a specific solution method and supports matrix types double and float.

Common Parameters:

- jobv — Determines whether to compute eigenvectors (EIGVALUES_V) or only eigenvalues (EIGVALUES_N).
- range — Specifies the range of computed eigenvalues (BLASRANGE_A, BLASRANGE_V, BLASRANGE_I).
- lower and upper — Lower and upper bounds for computing a subset of the spectrum.
- abstol — Absolute error tolerance.

All functions operate on symmetric tridiagonal matrices and allow selecting the most suitable algorithm depending on performance and accuracy requirements.

Function	Action
EigenTridiagonalDC	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEVD).
EigenTridiagonalQR	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEV).
EigenTridiagonalRobust	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack function STEVR).
EigenTridiagonalBisect	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the bisection algorithm (lapack function STEVX).
EigenTridiagonalQL	Compute all eigenvalues of a symmetric tridiagonal matrix using the square-root-free variant of the QL or QR algorithm (lapack function STERF).
EigenTridiagonalDCQ	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEDC).
EigenTridiagonalQRQ	Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEQR).
EigenTridiagonalPosDefQ	Compute eigenvalues and eigenvectors of a symmetric positive definite (положительно

Function	Action
	определённая) tridiagonal matrix using the QR algorithm (lapack function PTEQR).

EigenTridiagonalDC

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEVD).

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vector&                eigen_values,       // vector of computed eigenvalues
   matrix&                eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalDC(
   ENUM_EIG_VALUES        jobv,               // compute eigenvectors or not
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixf&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenTridiagonalQR

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEV).

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalQR(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   vector&                eigen_values,      // vector of computed eigenvalues
   matrix&                eigen_vectors      // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalQR(
   ENUM_EIG_VALUES        jobv,              // compute eigenvectors or not
   vectorf&               eigen_values,      // vector of computed eigenvalues
   matrixf&               eigen_vectors      // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the jobv parameter.

When jobv = EIGVALUES_V, eigenvectors and eigenvalues are calculated.

If EIGVALUES_N is set, eigenvectors are not calculated. Only eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIG_VALUES

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are calculated, without vectors.

EigenTridiagonalRobust

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Multiple Relatively Robust Representations, MRRR algorithm (lapack function STEVR).

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalRobust(
   ENUM_EIG_VALUES        jobv,                // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,               // subset of eigenvalues to compute
   double                 lower,               // lower bound of the subset
   double                 upper,               // Upper bound of the subset
   double                 abstol,              // absolute error tolerance
   vector&                eigen_values,        // vector of computed eigenvectors
   matrix&                eigen_vectors        // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalRobust(
   ENUM_EIG_VALUES        jobv,                // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,               // subset of eigenvalues to compute
   float                  lower,               // lower bound of the subset
   float                  upper,               // upper bound of the subset
   float                  abstol,              // absolute error tolerance
   vectorf&               eigen_values,        // vector of computed eigenvectors
   matrixf&               eigen_vectors        // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.

ID	Description
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenTridiagonalBisect

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the bisection algorithm (lapack function STEVX).

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalBisect(
   ENUM_EIG_VALUES        jobv,                 // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,                // subset of eigenvalues to compute
   double                 lower,                // lower bound of the subset
   double                 upper,                // Upper bound of the subset
   double                 abstol,               // absolute error tolerance
   vector&                eigen_values,         // vector of computed eigenvectors
   matrix&                eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalBisect(
   ENUM_EIG_VALUES        jobv,                 // compute eigenvectors or not
   ENUM_BLAS_RANGE        range,                // subset of eigenvalues to compute
   float                  lower,                // lower bound of the subset
   float                  upper,                // upper bound of the subset
   float                  abstol,               // absolute error tolerance
   vectorf&               eigen_values,         // vector of computed eigenvectors
   matrixf&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Parameters

jobv

[in] ENUM_EIG_VALUES enumeration value which determines the method for computing eigenvectors.

range

[in] ENUM_BLAS_RANGE enumeration value that defines a subset of computable eigenvalues and vectors.

lower

[in] The lower bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

upper

[in] The upper bound of eigenvalues subset; it is specified depending on the value of the '*range*' parameter.

abstol

[in] Absolute error tolerance.

The absolute error tolerance to which each eigenvalue/eigenvector is required.

If jobv = 'V', the eigenvalues and eigenvectors output have residual norms bounded by abstol, and the dot products between different eigenvectors are bounded by abstol.

If abstol < n *eps*|T|, then n *eps*|T| is used instead, where eps is the machine precision, and |T| is the 1-norm of the matrix T. The eigenvalues are computed to an accuracy of eps*|T| irrespective of abstol.

If high relative precision is important, 'abstol' should be set to a safe minimum value X such that 1.0/X does not overflow.

eigen_values

[out] Vector of eigenvalues.

V

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the values of the *jobv* and *range* parameters.

When BLASRANGE_A is set, all eigenvalues are computed, and the lower and upper parameters are ignored.

With the BLASRANGE_V value, only those eigenvalues (and their vectors) are computed, which fall within the range of real values specified by the 'lower' and 'upper' parameters.

With the BLASRANGE_I value, only those eigenvalues (and their vectors) are computed, which fall within the range of integer indices specified by the 'lower' and 'upper' parameters. For example, with lower=0 and upper=2, only the first three eigenvalues are computed.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIG_VALUES

An enumeration defining the need to compute eigenvectors.

ID	Description
EIGVALUES_V	Eigenvectors and eigenvalues are calculated.
EIGVALUES_N	Only eigenvalues are computed, without vectors.

ENUM_BLAS_RANGE

An enumeration defining how right singular vectors should be computed.

ID	Description
BLASRANGE_A	All singular or eigenvalues will be found.

ID	Description
BLASRANGE_V	All singular or eigenvalues in the half-open interval (VL,VU] will be found.
BLASRANGE_I	The IL-th through IU-th singular or eigenvalues will be found.

EigenTridiagonalQL

Compute all eigenvalues of a symmetric tridiagonal matrix using the square-root-free variant of the QL or QR algorithm (lapack function STERF).

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalQL(
   vector&              eigen_values        // vector of computed eigenvalues
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalQL(
   vectorf&             eigen_values        // vector of computed eigenvalues
   );
```

Parameters

eigen_values

[out] Vector of eigenvalues.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input must be a symmetric matrix in the tridiagonal form.

EigenTridiagonalDCQ

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer algorithm (lapack function STEDC). Unlike EigenTridiagonalDC, this method can be used to compute the eigenvectors of the original symmetric matrix. A symmetric matrix can be reduced to tridiagonal form using the ReduceSymmetricToTridiagonal method. The orthogonal matrix Q obtained from this transformation is then used to compute the eigenvectors of the original symmetric matrix.

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z      compv,              // compute eigenvectors or not
   matrix&                Q,                  // orthogonal matrix used in the reducti
   vector&                eigen_values,       // vector of computed eigenvalues
   matrix&                eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z      compv,              // compute eigenvectors or not
   matrixf&               Q,                  // orthogonal matrix used in the reducti
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixf&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z      compv,              // compute eigenvectors or not
   matrixc&               Q,                  // orthogonal matrix used in the reducti
   vector&                eigen_values,       // vector of computed eigenvalues
   matrixc&               eigen_vectors       // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenTridiagonalDCQ(
   ENUM_EIGTRIDIAG_Z      compv,              // compute eigenvectors or not
   matrixcf&              Q,                  // orthogonal matrix used in the reducti
   vectorf&               eigen_values,       // vector of computed eigenvalues
   matrixcf&              eigen_vectors       // matrix of computed eigenvectors
   );
```

Parameters

compv

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

Q

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIGTRIDIAG_Z

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGCOMPZ_N	'N': Compute eigenvalues only
EIGCOMPZ_V	'V': Compute eigenvectors of original symmetric matrix also
EIGCOMPZ_I	'I': Compute eigenvectors of tridiagonal matrix also

EigenTridiagonalQRQ

Compute eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm (lapack function STEQR). Unlike EigenTridiagonalQR, this method can be used to compute the eigenvectors of the original symmetric matrix. A symmetric matrix can be reduced to tridiagonal form using the ReduceSymmetricToTridiagonal method. The orthogonal matrix Q obtained from this transformation is then used to compute the eigenvectors of the original symmetric matrix.

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrix&                Q,                    // orthogonal matrix used in the reductio
   vector&                eigen_values,         // vector of computed eigenvalues
   matrix&                eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixf&               Q,                    // orthogonal matrix used in the reductio
   vectorf&               eigen_values,         // vector of computed eigenvalues
   matrixf&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixc&               Q,                    // orthogonal matrix used in the reductio
   vector&                eigen_values,         // vector of computed eigenvalues
   matrixc&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenTridiagonalQRQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixcf&              Q,                    // orthogonal matrix used in the reductio
   vectorf&               eigen_values,         // vector of computed eigenvalues
   matrixcf&              eigen_vectors         // matrix of computed eigenvectors
   );
```

Parameters

compv

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

Q

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric matrix in the tridiagonal form.

ENUM_EIGTRIDIAG_Z

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGCOMPZ_N	'N': Compute eigenvalues only
EIGCOMPZ_V	'V': Compute eigenvectors of original symmetric matrix also
EIGCOMPZ_I	'I': Compute eigenvectors of tridiagonal matrix also

EigenTridiagonalPosDefQ

Compute eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix using the QR algorithm (lapack function PTEQR). This method can be used to compute the eigenvectors of both the input tridiagonal matrix and the original positive definite symmetric matrix. This is achieved using the orthogonal matrix Q obtained during the reduction to tridiagonal form.

Computing for type matrix<double>

```
bool  matrix::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrix&                Q,                    // orthogonal matrix used in the reductio
   vector&                eigen_values,         // vector of computed eigenvalues
   matrix&                eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixf&               Q,                    // orthogonal matrix used in the reductio
   vectorf&               eigen_values,         // vector of computed eigenvalues
   matrixf&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixc&               Q,                    // orthogonal matrix used in the reductio
   vector&                eigen_values,         // vector of computed eigenvalues
   matrixc&               eigen_vectors         // matrix of computed eigenvectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::EigenTridiagonalPosDefQ(
   ENUM_EIGTRIDIAG_Z      compv,                // compute eigenvectors or not
   matrixcf&              Q,                    // orthogonal matrix used in the reductio
   vectorf&               eigen_values,         // vector of computed eigenvalues
   matrixcf&              eigen_vectors         // matrix of computed eigenvectors
   );
```

Parameters

compv

[in] ENUM_EIGTRIDIAG_Z enumeration value which determines the method for computing eigenvectors.

Q

[in] Orthogonal matrix Q produced by method ReflectTridiagonalToQ.

eigen_values

[out] Vector of eigenvalues.

eigen_vectors

[out] Matrix of eigenvectors.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Computation depends on the value of the compv parameter.

When compv = EIGCOMPZ_N, compute eigenvalues only, eigenvectors are not calculated.

If EIGCOMPZ_V is set, eigenvalues are computed and eigenvectors of original symmetric matrix are calculated also.

If EIGCOMPZ_I is set, eigenvalues are computed and eigenvectors of tridiagonal matrix are calculated also.

The input must be a symmetric positive definite tridiagonal matrix.

ENUM_EIGTRIDIAG_Z

An enumeration that specifies whether to calculate eigenvectors.

ID	Description
EIGCOMPZ_N	'N': Compute eigenvalues only
EIGCOMPZ_V	'V': Compute eigenvectors of original symmetric matrix also
EIGCOMPZ_I	'I': Compute eigenvectors of tridiagonal matrix also

Orthogonal Factorizations

OpenBLAS provides a number of routines for factorizing a general rectangular m-by-n matrix A, as the product of an orthogonal matrix (unitary if complex) and a triangular (or possibly trapezoidal) matrix.

A real matrix Q is orthogonal if $Q^T Q = I$; a complex matrix Q is unitary if $Q^H Q = I$. Orthogonal or unitary matrices have the important property that they leave the two-norm of a vector invariant:

$||x||_2 = ||Qx||_2$, if Q is orthogonal or unitary

As a result, they help to maintain numerical stability because they do not amplify rounding errors.

Orthogonal factorizations are used in the solution of linear least squares problems. They may also be used to perform preliminary steps in the solution of eigenvalue or singular value problems.

Function	Action
FactorizationQR	Computes the QR factorization of a general m-by-n matrix: A = Q * R. Lapack function GEQRF.
FactorizationQRNonNeg	Computes the QR factorization of a general m-by-n matrix: A = Q * R. R is an upper triangular matrix with nonnegative diagonal entries. Lapack function GEQRFP.
FactorizationQRPivot	Computes the QR factorization of a general m-by-n matrix with column pivoting: A * P = Q * R. Lapack function GEQP3.
FactorizationLQ	Computes the LQ factorization of a general m-by-n matrix: A = L * Q. Lapack function GELQF.
FactorizationQL	Computes the QL factorization of a general m-by-n matrix: A = Q * L. Lapack function GEQLF.
FactorizationRQ	Computes the RQ factorization of a general m-by-n matrix: A = R * Q. Lapack function GERQF.

FactorizationQR

Computes the QR factorization of a general m-by-n matrix: A = Q * R. Lapack function GEQRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationQR(
   bool              reduced,       // calculation mode reduced or complete
   matrix&           Q,             // orthogonal matrix Q
   matrix&           R              // upper triangular matrix R
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationQR(
   bool              reduced,       // calculation mode reduced or complete
   matrixf&          Q,             // orthogonal matrix Q
   matrixf&          R              // upper triangular matrix R
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationQR(
   bool              reduced,       // calculation mode reduced or complete
   matrixc&          Q,             // unitary matrix Q
   matrixc&          R              // upper triangular matrix R
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationQR(
   bool              reduced,       // calculation mode reduced or complete
   matrixcf&         Q,             // unitary matrix Q
   matrixcf&         R              // upper triangular matrix R
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

Q

[out] Orthogonal or unitary matrix Q.

R

[out] Upper triangular matrix R.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If `reduced` is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If `reduced` is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

FactorizationQRNonNeg

Computes the QR factorization of a general m-by-n matrix: A = Q * R. R is an upper triangular matrix with nonnegative diagonal entries. Lapack function GEQRFP.

Computing for type matrix<double>

```
bool  matrix::FactorizationQRNonNeg(
   bool            reduced,       // calculation mode reduced or complete
   matrix&         Q,             // orthogonal matrix Q
   matrix&         R              // upper triangular matrix R
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationQRNonNeg(
   bool            reduced,       // calculation mode reduced or complete
   matrixf&        Q,             // orthogonal matrix Q
   matrixf&        R              // upper triangular matrix R
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationQRNonNeg(
   bool            reduced,       // calculation mode reduced or complete
   matrixc&        Q,             // unitary matrix Q
   matrixc&        R              // upper triangular matrix R
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationQRNonNeg(
   bool            reduced,       // calculation mode reduced or complete
   matrixcf&       Q,             // unitary matrix Q
   matrixcf&       R              // upper triangular matrix R
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

Q

[out] Orthogonal or unitary matrix Q.

R

[out] Upper triangular matrix R.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

FactorizationQRPivot

Computes the QR factorization of a general m-by-n matrix with column pivoting: A * P = Q * R. Lapack function GEQP3.

Computing for type matrix<double>

```
bool  matrix::FactorizationQRPivot(
   bool            reduced,       // calculation mode reduced or complete
   long[]&         jpvt,          // array with predefined permutations
   matrix&         Q,             // orthogonal matrix Q
   matrix&         R,             // upper triangular matrix R
   matrix&         P              // permutation matrix
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationQRPivot(
   bool            reduced,       // calculation mode reduced or complete
   long[]&         jpvt,          // array with predefined permutations
   matrixf&        Q,             // orthogonal matrix Q
   matrixf&        R,             // upper triangular matrix R
   matrixf&        P              // permutation matrix
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationQRPivot(
   bool            reduced,       // calculation mode reduced or complete
   long[]&         jpvt,          // array with predefined permutations
   matrixc&        Q,             // unitary matrix Q
   matrixc&        R,             // upper triangular matrix R
   matrixc&        P              // permutation matrix
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationQRPivot(
   bool            reduced,       // calculation mode reduced or complete
   long[]&         jpvt,          // array with predefined permutations
   matrixcf&       Q,             // unitary matrix Q
   matrixcf&       R,             // upper triangular matrix R
   matrixcf&       P              // permutation matrix
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices Q, R calculated with dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, R with dimensions (M,M), (M,N).

jpvt

[in] Integer array of dimension n. if jpvt(i) ≠ 0, the i-th column of A is moved to the beginning of AP before the computation, and fixed in place during the computation. If jpvt(i) = 0, the i-th column of A is a free column (that is, it may be interchanged during the computation with any other free column). If array has zero size (or not initialized), then all the columns of A assumed to be free.

Q

[out] Orthogonal or unitary matrix Q.

R

[out] Upper triangular matrix R.

P

[out] Permutation matrix P of n-by-n sizes.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If *reduced* is true

If m >= n, matrix Q is of m-by-n sizes, matrix R is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

If *reduced* is false, matrix Q is of m-by-m sizes, matrix R is of m-by-n sizes.

FactorizationLQ

Computes the LQ factorization of a general m-by-n matrix: A = L * Q. Lapack function GELQF.

Computing for type matrix<double>

```
bool  matrix::FactorizationLQ(
   bool             reduced,       // calculation mode reduced or complete
   matrix&          L,             // lower triangular matrix L
   matrix&          Q              // orthogonal matrix Q
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationLQ(
   bool             reduced,       // calculation mode reduced or complete
   matrixf&         L,             // lower triangular matrix L
   matrixf&         Q              // orthogonal matrix Q
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationLQ(
   bool             reduced,       // calculation mode reduced or complete
   matrixc&         L,             // lower triangular matrix L
   matrixc&         Q              // unitary matrix Q
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationLQ(
   bool             reduced,       // calculation mode reduced or complete
   matrixcf&        L,             // lower triangular matrix L
   matrixcf&        Q              // unitary matrix Q
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices L, Q calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices L, Q with dimensions (M,N), (N,N).

L

[out] Lower triangular matrix L.

Q

[out] Orthogonal or unitary matrix Q.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If `reduced` is true

If m <= n, matrix L is of m-by-m sizes, matrix Q is of m-by-n sizes.

If m > n, matrix L is of m-by-n sizes, matrix Q is of n-by-n sizes.

If `reduced` is false, matrix L is of m-by-n sizes, matrix Q is of n-by-n sizes.

FactorizationQL

Computes the QL factorization of a general m-by-n matrix: A = Q * L. Lapack function GEQLF.

Computing for type matrix<double>

```
bool  matrix::FactorizationQL(
   bool              reduced,       // calculation mode reduced or complete
   matrix&           Q,             // orthogonal matrix Q
   matrix&           L              // lower triangular matrix L
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationQL(
   bool              reduced,       // calculation mode reduced or complete
   matrixf&          Q,             // orthogonal matrix Q
   matrixf&          L              // lower triangular matrix L
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationQL(
   bool              reduced,       // calculation mode reduced or complete
   matrixc&          Q,             // unitary matrix Q
   matrixc&          L              // lower triangular matrix L
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationQL(
   bool              reduced,       // calculation mode reduced or complete
   matrixcf&         Q,             // unitary matrix Q
   matrixcf&         L              // lower triangular matrix L
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices Q, L calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices Q, L with dimensions (M,M), (M,N).

Q

[out] Orthogonal or unitary matrix Q.

L

[out] Lower triangular matrix L.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If *`reduced`* is true

If m >= n, matrix Q is of m-by-n sizes, matrix L is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix L is of m-by-n sizes.

If *`reduced`* is false, matrix Q is of m-by-m sizes, matrix L is of m-by-n sizes.

FactorizationRQ

Computes the RQ factorization of a general m-by-n matrix: A = R * Q. Lapack function GERQF.

Computing for type matrix<double>

```
bool  matrix::FactorizationRQ(
   bool              reduced,       // calculation mode reduced or complete
   matrix&           R,             // upper triangular matrix R
   matrix&           Q              // orthogonal matrix Q
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationRQ(
   bool              reduced,       // calculation mode reduced or complete
   matrixf&          R,             // upper triangular matrix R
   matrixf&          Q              // orthogonal matrix Q
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationRQ(
   bool              reduced,       // calculation mode reduced or complete
   matrixc&          R,             // upper triangular matrix R
   matrixc&          Q              // unitary matrix Q
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationRQ(
   bool              reduced,       // calculation mode reduced or complete
   matrixcf&         R,             // upper triangular matrix R
   matrixcf&         Q              // unitary matrix Q
   );
```

Parameters

reduced

[in] Calculation mode. If reduced is true then matrices R, Q calculated with reduced dimensions (M, K), (K, N). If reduced is false it means complete calculation of matrices R, Q with dimensions (M,N), (N,N).

R

[out] Upper triangular matrix R.

Q

[out] Orthogonal or unitary matrix Q.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If *reduced* is true

If m <= n, matrix R is of m-by-m sizes, matrix Q is of m-by-n sizes.

If m > n, matrix R is of m-by-n sizes, matrix Q is of n-by-n sizes.

If *reduced* is false, matrix R is of m-by-n sizes, matrix Q is of n-by-n sizes.

Linear Equations

This section provides functions for solving systems of linear equations of the form A·X = B, where A is a square coefficient matrix, B is the right-hand side, and X is the solution matrix or vector. The methods support various data types (double, float, complex, complexf) and are based on LAPACK functions to ensure high precision and performance.

The functions are categorized based on the type of the coefficient matrix A. Each function is overloaded to support both matrix and vector forms for the right-hand side (B) and the solution (X).

Function	Action
LinearEquationsSolution	Computes the solution to the system of linear equations with a square coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GESV.
LinearEquationsSolutionTriangular	Computes the solution to the system of linear equations with a square triangular coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function TRTRS.
LinearEquationsSolutionSy	Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack functions SYSV, HESV.
LinearEquationsSolutionComplexSy	Computes the solution to the system of linear equations with a complex symmetric (not Hermitian conjugated!) matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n complex symmetric matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function SYSV.
LinearEquationsSolutionSyPD	Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated positive-definite matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated positive-definite matrix, the columns of matrix B are individual right-hand sides,

Function	Action
	and the columns of X are the corresponding solutions. Lapack function POSV.
LinearEquationsSolutionGeTrid	Computes the solution to the system of linear equations with a general (nonsymmetric) tridiagonal coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GTSV.
LinearEquationsSolutionSyTridPD	Computes the solution to the system of linear equations with a symmetric tridiagonal positive-definite coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function PTSV.

LinearEquationsSolution

Computes the solution to the system of linear equations with a square coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GESV.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolution(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolution(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolution(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolution(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolution(
    vectorcf&       B,              // right hand side vector B
    vectorcf&       X               // solution vector X
    );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

LinearEquationsSolutionTriangular

Computes the solution to the system of linear equations with a square triangular coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function TRTRS.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   matrix&             B,             // right hand side matrix B
   matrix&             X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   vector&             B,             // right hand side vector B
   vector&             X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   matrixf&            B,             // right hand side matrix B
   matrixf&            X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   vectorf&            B,             // right hand side vector B
   vectorf&            X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   matrixc&            B,             // right hand side matrix B
   matrixc&            X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   vectorc&            B,             // right hand side vector B
   vectorc&            X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   matrixcf&           B,             // right hand side matrix B
   matrixcf&           X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionTriangular(
   ENUM_EQUATIONS_FORM trans,         // form of the system of equations
   vectorcf&           B,             // right hand side vector B
   vectorcf&           X              // solution vector X
   );
```

Parameters

trans

[in] ENUM_EQUATIONS_FORM enumeration value which specifies the form of the system of equations.

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

ENUM_EQUATIONS_FORM

An enumeration defining which form of the equations' system calculated.

ID	Description
EQUATIONSFORM_N	'N': A * X = B (No transpose)
EQUATIONSFORM_T	'T': A**T * X = B (Transpose)
EQUATIONSFORM_C	'C': A**H * X = B (Conjugate transpose)

In case of real matrices the value EQUATIONSFORM_C assumed as Transpose.

LinearEquationsSolutionSy

Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack functions SYSV, HESV.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionSy(
   matrix&         B,             // right hand side matrix B
   matrix&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vector&         B,             // right hand side vector B
   vector&         X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionSy(
   matrixf&        B,             // right hand side matrix B
   matrixf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vectorf&         B,             // right hand side vector B
   vectorf&         X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionSy(
   matrixc&        B,             // right hand side matrix B
   matrixc&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vectorc&        B,             // right hand side vector B
   vectorc&        X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionSy(
   matrixcf&       B,             // right hand side matrix B
   matrixcf&       X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSy(
   vectorcf&        B,             // right hand side vector B
   vectorcf&        X              // solution vector X
   );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a symmetric (Hermitian conjugated), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

LinearEquationsSolutionComplexSy

Computes the solution to the system of linear equations with a complex symmetric (not Hermitian conjugated!) matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n complex symmetric matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function SYSV.

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionComplexSy(
   matrixc&         B,              // right hand side matrix B
   matrixc&         X               // solution matrix X
   );

bool  matrix::LinearEquationsSolutionComplexSy(
   vectorc&         B,              // right hand side vector B
   vectorc&         X               // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionComplexSy(
   matrixcf&        B,              // right hand side matrix B
   matrixcf&        X               // solution matrix X
   );

bool  matrix::LinearEquationsSolutionComplexSy(
   vectorcf&        B,              // right hand side vector B
   vectorcf&        X               // solution vector X
   );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a complex symmetric, upper triangular or lower triangular matrix. Triangular matrices are assumed to be complex symmetric.

LinearEquationsSolutionSyPD

Computes the solution to the system of linear equations with a symmetric or Hermitian conjugated positive-definite matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric or Hermitian conjugated positive-definite matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function POSV.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrix&         B,             // right hand side matrix B
   matrix&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vector&         B,             // right hand side vector B
   vector&         X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixf&        B,             // right hand side matrix B
   matrixf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vectorf&         B,             // right hand side vector B
   vectorf&         X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixc&        B,             // right hand side matrix B
   matrixc&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vectorc&        B,             // right hand side vector B
   vectorc&        X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionSyPD(
   matrixcf&       B,             // right hand side matrix B
   matrixcf&       X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyPD(
   vectorcf&        B,              // right hand side vector B
   vectorcf&        X               // solution vector X
   );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

The input can be a symmetric (Hermitian conjugated), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

LinearEquationsSolutionGeTrid

Computes the solution to the system of linear equations with a general (nonsymmetric) tridiagonal coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n general tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function GTSV.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrix&         B,             // right hand side matrix B
   matrix&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vector&         B,             // right hand side vector B
   vector&         X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixf&        B,             // right hand side matrix B
   matrixf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vectorf&         B,             // right hand side vector B
   vectorf&         X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixc&        B,             // right hand side matrix B
   matrixc&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vectorc&        B,             // right hand side vector B
   vectorc&        X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionGeTrid(
   matrixcf&       B,             // right hand side matrix B
   matrixcf&       X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionGeTrid(
   vectorcf&        B,             // right hand side vector B
   vectorcf&        X              // solution vector X
   );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

LinearEquationsSolutionSyTridPD

Computes the solution to the system of linear equations with a symmetric tridiagonal positive-definite coefficient matrix A and multiple right-hand sides. A*X = B, where A is an n-by-n symmetric tridiagonal matrix, the columns of matrix B are individual right-hand sides, and the columns of X are the corresponding solutions. Lapack function PTSV.

Computing for type matrix<double>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrix&          B,             // right hand side matrix B
   matrix&          X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vector&          B,             // right hand side vector B
   vector&          X              // solution vector X
   );
```

Computing for type matrix<float>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixf&         B,             // right hand side matrix B
   matrixf&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vectorf&          B,             // right hand side vector B
   vectorf&          X              // solution vector X
   );
```

Computing for type matrix<complex>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixc&         B,             // right hand side matrix B
   matrixc&         X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
   vectorc&         B,             // right hand side vector B
   vectorc&         X              // solution vector X
   );
```

Computing for type matrix<complexf>

```
bool  matrix::LinearEquationsSolutionSyTridPD(
   matrixcf&        B,             // right hand side matrix B
   matrixcf&        X              // solution matrix X
   );

bool  matrix::LinearEquationsSolutionSyTridPD(
    vectorcf&        B,             // right hand side vector B
    vectorcf&        X              // solution vector X
    );
```

Parameters

B

[in] Matrix B whose columns are the right-hand sides for the systems of equations. Vector B contains one column of right-hand side.

X

[out] Matrix or vector X with solutions of linear equations system.

Return Value

Return true if successful, otherwise false in case of an error.

Note

Output matrix X has the same sizes as input matrix B. Output vector X has the same size as input vector B.

Factorizations

The Factorizations section contains functions for performing various types of matrix factorizations used in numerical solutions of linear systems, stability analysis, and other linear algebra tasks. These factorizations transform the original matrix into simpler forms, making subsequent computations more efficient. All functions are implemented using LAPACK routines and support the types double, float, complex, and complexf.

The functions in this section are used for:

- Preprocessing matrices when solving systems of linear equations;
- Computing determinants, ranks, and matrix inverses;
- Assessing the stability of numerical methods;
- Solving problems in spectral theory and optimization methods.

Matrix factorization is a critical step in many linear algebra algorithms, and this section provides access to the most efficient and well-established factorization techniques.

Function	Action
FactorizationPLU	Computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges. The factorization has the form A = P * L * U. Lapack function GETRF.
FactorizationPLUGeTrid	Computes an LU factorization of a general (non-symmetric) tridiagonal N-by-N matrix A using elimination with partial pivoting and row interchanges. The factorization has the form A = P * L * U. Lapack function GTTRF.
FactorizationLDL	Computes the factorization of a real symmetric or complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. Lapack functions SYTRF, HETRF.
FactorizationLDLSyTridPD	Forms the factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite tridiagonal matrix A. Lapack function PTTRF.
FactorizationCholesky	Computes the factorization of a real symmetric or complex Hermitian positive-definite matrix A. Lapack function POTRF.
FactorizationCholeskySyPS	Computes the Cholesky factorization with complete pivoting of a real symmetric (complex Hermitian) positive semidefinite N-by-N matrix. Lapack function PSTRF.

FactorizationPLU

Computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges. The factorization has the form

A = P * L * U

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n). Lapack function GETRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationPLU(
   matrix&          P,             // permutation matrix P
   matrix&          L,             // lower triangular matrix L
   matrix&          U              // upper triangular matrix U
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationPLU(
   matrixf&         P,             // permutation matrix P
   matrixf&         L,             // lower triangular matrix L
   matrixf&         U              // upper triangular matrix U
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationPLU(
   matrixc&         P,             // permutation matrix P
   matrixc&         L,             // lower triangular matrix L
   matrixc&         U              // upper triangular matrix U
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationPLU(
   matrixcf&        P,             // permutation matrix P
   matrixcf&        L,             // lower triangular matrix L
   matrixcf&        U              // upper triangular matrix U
   );
```

Parameters

P

[out] Permutation matrix P.

L

[out] Lower triangular matrix L with unit diagonal elements.

U

[out] Upper triangular matrix U.

Return Value

Return true if successful, otherwise false in case of an error.

FactorizationPLUGeTrid

Computes an LU factorization of a general (non-symmetric) tridiagonal N-by-N matrix A using elimination with partial pivoting and row interchanges. The factorization has the form

A = P * L * U

where P is a permutation matrix, L is lower triangular with unit diagonal elements, and U is upper triangular. Lapack function GTTRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationPLUGeTrid(
   matrix&          P,              // permutation matrix P
   matrix&          L,              // lower triangular matrix L
   matrix&          U               // upper triangular matrix U
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationPLUGeTrid(
   matrixf&         P,              // permutation matrix P
   matrixf&         L,              // lower triangular matrix L
   matrixf&         U               // upper triangular matrix U
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationPLUGeTrid(
   matrixc&         P,              // permutation matrix P
   matrixc&         L,              // lower triangular matrix L
   matrixc&         U               // upper triangular matrix U
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationPLUGeTrid(
   matrixcf&        P,              // permutation matrix P
   matrixcf&        L,              // lower triangular matrix L
   matrixcf&        U               // upper triangular matrix U
   );
```

Parameters

P

[out] Permutation matrix P.

L

[out] Lower triangular matrix L with unit diagonal elements.

U

[out] Upper triangular matrix U.

Return Value

Return true if successful, otherwise false in case of an error.

FactorizationLDL

Computes the factorization of a real symmetric or complex Hermitian matrix A using the Bunch-Kaufman diagonal pivoting method. The form of the factorization is:

A = L * D * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * D * U in case of upper triangular matrix A

where L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements. D is a symmetric block-diagonal matrix with 1-by-1 and 2-by-2 diagonal blocks. Lapack functions SYTRF, HETRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationLDL(
   matrix&          L,             // lower or upper triangular matrix
   matrix&          D              // diagonal matrix D
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationLDL(
   matrixf&         L,             // lower or upper triangular matrix
   matrixf&         D              // diagonal matrix D
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationLDL(
   matrixc&         L,             // lower or upper triangular matrix
   matrixc&         D              // diagonal matrix D
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationLDL(
   matrixcf&        L,             // lower or upper triangular matrix
   matrixcf&        D              // diagonal matrix D
   );
```

Parameters

L

[out] Lower or upper triangular matrix with unit diagonal elements.

D

[out] Symmetric block-diagonal matrix D.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

FactorizationLDLSyTridPD

Forms the factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite tridiagonal matrix A:

A = L * D * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * D * U in case of upper triangular matrix A

where L is lower triangular with unit diagonal elements, U is upper triangular with unit diagonal elements. D is diagonal matrix. Lapack function PTTRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationLDLSyTridPD(
   matrix&          L,             // lower or upper triangular matrix
   matrix&          D              // diagonal matrix D
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixf&         L,             // lower or upper triangular matrix
   matrixf&         D              // diagonal matrix D
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixc&         L,             // lower or upper triangular matrix
   matrixc&         D              // diagonal matrix D
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationLDLSyTridPD(
   matrixcf&        L,             // lower or upper triangular matrix
   matrixcf&        D              // diagonal matrix D
   );
```

Parameters

L

[out] Lower or upper triangular matrix with unit diagonal elements.

D

[out] Diagonal matrix D.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

FactorizationCholesky

Computes the factorization of a real symmetric or complex Hermitian positive-definite matrix A. The factorization has the form:

A = L * L**T in case of lower triangular or symmetric matrix A

or

A = U**T * U in case of upper triangular matrix A

where L is lower triangular, U is upper triangular. Lapack function POTRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationCholesky(
   matrix&          L              // lower or upper triangular matrix
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationCholesky(
   matrixf&         L              // lower or upper triangular matrix
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationCholesky(
   matrixc&         L              // lower or upper triangular matrix
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationCholesky(
   matrixcf&        L              // lower or upper triangular matrix
   );
```

Parameters

L

[out] Lower or upper triangular matrix.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

FactorizationCholeskySyPS

Computes the Cholesky factorization with complete pivoting of a real symmetric (complex Hermitian) positive semidefinite N-by-N matrix. The form of the factorization is:

P**T * A * P = L * L**T in case of lower triangular or symmetric matrix A

or

P**T * A * P = U**T * U in case of upper triangular matrix A

where P is a permutation matrix, L is lower triangular, U is upper triangular. Lapack function PSTRF.

Computing for type matrix<double>

```
bool  matrix::FactorizationCholeskySyPS(
   double          tol            // tolerance
   matrix&         P,             // permutation matrix P
   matrix&         L              // lower or upper triangular matrix
   );
```

Computing for type matrix<float>

```
bool  matrix::FactorizationCholeskySyPS(
   float           tol            // tolerance
   matrixf&        P,             // permutation matrix P
   matrixf&        L              // lower or upper triangular matrix
   );
```

Computing for type matrix<complex>

```
bool  matrix::FactorizationCholeskySyPS(
   double          tol            // tolerance
   matrixc&        P,             // permutation matrix P
   matrixc&        L              // lower or upper triangular matrix
   );
```

Computing for type matrix<complexf>

```
bool  matrix::FactorizationCholeskySyPS(
   float           tol            // tolerance
   matrixcf&       P,             // permutation matrix P
   matrixcf&       L              // lower or upper triangular matrix
   );
```

Parameters

tol

[in] User defined tolerance. If tol < 0, then n*ε*max(A[k,k]), where ε is the machine precision, will be used. The algorithm terminates at the (k-1)st step, if the pivot <=tol.

P

[out] Permutation matrix P.

L

[out] Lower or upper triangular matrix.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The input can be a symmetric (Hermitian), upper triangular or lower triangular matrix. Triangular matrices are assumed to be symmetric (Hermitian conjugated).

Dynamic Mode Decomposition

The Dynamic Mode Decomposition (DMD) functions are designed to compute an evolution operator from data snapshots, enabling efficient analysis of linear dynamics in extended phase spaces. The methods provide memory-efficient representations of evolution operators, with DynamicModeDecompositionQR incorporating QR-based data compression for improved computational efficiency.

Function	Action
DynamicModeDecomposition	Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices. Lapack function GEDMD.
DynamicModeDecompositionQR	Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices, using a QR factorization based compression of the data. Lapack function GEDMDQ.

The Dynamic mode decomposition (DMD) method is an algorithm for searching for an evolution operator (inverse operator problem solutions) in a finite-dimensional problem solution space (numerical or experimentally obtained) in a set of solutions (slices, "snapshots") in some consecutive moments of time. Expansion of the phase space due to the use of a nonlinear basis (relative to the variables of the problem) allows us to construct a global linear operator describing a linear evolution in the extended "rectifying space" (the Coopman operator) and the Perron-Frobenius operator that is its adjoint one. The DMD method is equivalent to a compressed representation of a linear evolution operator in the form of a product of rectangular matrices, which provides significant savings in the required memory during calculations.

DynamicModeDecomposition

Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices. Lapack function GEDMD. For the input matrices X and Y such that Y = A*X with an unaccessible matrix A, GEDMD computes a certain number of Ritz pairs of A using the standard Rayleigh-Ritz extraction from a subspace of range(X) that is determined using the leading left singular vectors of X. Optionally, GEDMD returns the residuals of the computed Ritz pairs, the information needed for a refinement of the Ritz vectors, or the eigenvectors of the Exact DMD. Both matrices must be the same size.

Computing for type matrix<double>

```
bool  matrix::DynamicModeDecomposition(
   matrix&              B,                       // second snapshot matrix in the pa
   ENUM_DMD_SCALE       jobs,                    // determines whether the initial da
   ENUM_DMD_EIGV        jobz,                    // determines whether the eigenvect
   ENUM_DMD_RESIDUALS   jobr,                    // determines whether to compute the
   ENUM_DMD_REFINE      jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG         whtsvd,                  // allows for a selection of the SV
   long                 nrnk,                    // determines the mode how to comput
   double               tol,                     // the tolerance for truncating smal
   vectorc&             eigen_values,            // vector of eigenvalues (Ritz value
   matrix&              left_vectors,            // matrix of left singular vectors
   matrix&              Z                        // matrix of Ritz vectors
   vector&              residuals,               // residuals for the Ritz pairs
   matrix&              res_vectors,             // residual vectors for the Ritz pa
   matrix&              B,                       // computed vectors
   matrix&              W,                       // eigenvectors of the matrix Rayle
   matrix&              S                        // computed vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::DynamicModeDecomposition(
   matrixf&             B,                       // second snapshot matrix in the pa
   ENUM_DMD_SCALE       jobs,                    // determines whether the initial da
   ENUM_DMD_EIGV        jobz,                    // determines whether the eigenvect
   ENUM_DMD_RESIDUALS   jobr,                    // determines whether to compute the
   ENUM_DMD_REFINE      jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG         whtsvd,                  // allows for a selection of the SV
   long                 nrnk,                    // determines the mode how to comput
   float                tol,                     // the tolerance for truncating smal
   vectorcf&            eigen_values,            // vector of eigenvalues (Ritz value
   matrixf&             left_vectors,            // matrix of left singular vectors
   matrixf&             Z                        // matrix of Ritz vectors
   vectorf&             residuals,               // residuals for the Ritz pairs
   matrixf&             res_vectors,             // residual vectors for the Ritz pa
   matrixf&             B,                       // computed vectors
   matrixf&             W,                       // eigenvectors of the matrix Rayle
   matrixf&             S                        // computed vectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::DynamicModeDecomposition(
   matrixc&              B,                       // second snapshot matrix in the pa
   ENUM_DMD_SCALE        jobs,                    // determines whether the initial da
   ENUM_DMD_EIGV         jobz,                    // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                    // determines whether to compute the
   ENUM_DMD_REFINE       jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG          whtsvd,                  // allows for a selection of the SV
   long                  nrnk,                    // determines the mode how to comput
   double                tol,                     // the tolerance for truncating sma
   vectorc&              eigen_values,            // vector of eigenvalues (Ritz value
   matrixc&              left_vectors,            // matrix of left singular vectors
   matrixc&              Z                        // matrix of Ritz vectors
   vector&               residuals,               // residuals for the Ritz pairs
   matrixc&              res_vectors,             // residual vectors for the Ritz pa
   matrixc&              B,                       // computed vectors
   matrixc&              W,                       // eigenvectors of the matrix Rayle
   matrixc&              S                        // computed vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::DynamicModeDecomposition(
   matrixcf&             B,                       // second snapshot matrix in the pa
   ENUM_DMD_SCALE        jobs,                    // determines whether the initial da
   ENUM_DMD_EIGV         jobz,                    // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                    // determines whether to compute the
   ENUM_DMD_REFINE       jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG          whtsvd,                  // allows for a selection of the SV
   long                  nrnk,                    // determines the mode how to comput
   float                 tol,                     // the tolerance for truncating sma
   vectorcf&             eigen_values,            // vector of eigenvalues (Ritz value
   matrixcf&             left_vectors,            // matrix of left singular vectors
   matrixcf&             Z                        // matrix of Ritz vectors
   vectorf&              residuals,               // residuals for the Ritz pairs
   matrixcf&             res_vectors,             // residual vectors for the Ritz pa
   matrixcf&             B,                       // computed vectors
   matrixcf&             W,                       // eigenvectors of the matrix Rayle
   matrixcf&             S                        // computed vectors
   );
```

Parameters

Y

[in] The second snapshot matrix in the pair.

jobs

[in] Value from the ENUM_DMD_SCALE enumeration which determines whether the initial data snapshots are scaled by a diagonal matrix.

jobz

[in] Value from the ENUM_DMD_EIGV enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

jobr

[in] Value from the ENUM_DMD_RESIDUALS enumeration which determines whether to compute the residuals.

jobf

[in] Value from the ENUM_DMD_REFINE enumeration which specifies whether to store information needed for post-processing (e.g. computing refined Ritz vectors).

whtsvd

[in] Value from the ENUM_SVD_ALG enumeration which allows for a selection of the SVD algorithm from the LAPACK library.

nrnk

[in] Value determines the mode how to compute the numerical rank, i.e. how to truncate small singular values of the input matrix X. If

NRNK = -1 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(1). This option is recommended.

NRNK = -2 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(i-1). This option is included for R&D purposes. It requires highly accurate SVD, which may not be feasible.

The numerical rank can be enforced by using positive value of NRNK as follows: 0 < NRNK <= N :: at most NRNK largest singular values will be used. If the number of the computed nonzero singular values is less than NRNK, then only those nonzero values will be used and the actually used dimension is less than NRNK.

tol

[in] The tolerance for truncating small singular values. 0 <= TOL < 1

eigen_values

[out] Vector of eigenvalues of size K. The leading K (K<=N) entries of EIGS contain the computed eigenvalues (Ritz values).

left_vectors

[out] Matrix of left singular vectors. The leading K columns contain a POD basis (POD - Proper Orthogonal Decomposition, правильная ортогональная декомпозиция), i.e. the leading K left singular vectors of the input data matrix X All N columns of contain all left singular vectors of the input matrix X.

Z

[out] Matrix of Ritz vectors. If JOBZ =='V' then Z contains the Ritz vectors. Z(:,i) is an eigenvector of the i-th Ritz value; ||Z(:,i)||_2=1. If JOBZ == 'F', then the Z(:,i)'s are given implicitly as the columns of X(:,1:K)*W(1:K,1:K), i.e. X(:,1:K)*W(:,i) is an eigenvector corresponding to EIGS(i). The columns of W(1:k,1:K) are the computed eigenvectors of the K-by-K Rayleigh quotient.

residuals

[out] Residuals for the K computed Ritz pairs.

res_vectors

[out] Residual vectors for the K computed Ritz pairs.

B

[out] M-by-K matrix (K<=N).

If JOBF =='R', B(1:M,1:K) contains A*U(:,1:K), and can be used for computing the refined vectors; see further details in the provided references.

If JOBF == 'E', B(1:M,1:K) contains A*U(:,1:K)*W(1:K,1:K), which are the vectors from the Exact DMD, up to scaling by the inverse eigenvalues.

If JOBF =='N', then B is not referenced.

W

[out] K-by-K matrix (K<=N). Contains the K computed eigenvectors of the matrix Rayleigh quotient. The Ritz vectors (returned in Z) are the product of X (containing a POD basis for the input matrix X) and W.

S

[out] K-by-K matrix (K<=N). The array S(1:K,1:K) is used for the matrix Rayleigh quotient. This content is overwritten during the eigenvalue decomposition by GEEV.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The number of matrices rows (M) must not be less than the number of columns (N).

ENUM_DMD_SCALE

An enumeration that determines whether the initial data snapshots are scaled by a diagonal matrix.

ID	Description
DMDSCALE_S	'S': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that X*D has unit nonzero columns
DMDSCALE_C	'C': The snapshots are scaled as with the 'S' option.If it is found that an i-th column of X is zero vector and the corresponding i-th column of Y is non-zero, then the i-th column of Y is set to zero
DMDSCALE_Y	'Y': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that Y*D has unit nonzero columns
DMDSCALE_N	'N': No data scaling

ENUM_DMD_EIGV

An enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

ID	Description
DMDEIGV_V	'V': The eigenvectors (Koopman modes) will be computed
DMDEIGV_F	'F': The eigenvectors (Koopman modes) will be returned in factored form
DMDEIGV_N	'N': The eigenvectors are not computed

ENUM_DMD_RESIDUALS

An enumeration which determines whether to compute the residuals.

ID	Description
DMDRESIDUALS_R	'R': The residuals for the computed eigenpairs will be computed
DMDRESIDUALS_N	'N': The residuals are not computed

ENUM_DMD_REFINE

An enumeration which specifies whether to store information needed for post-processing.

ID	Description
DMDREFINE_R	'R': The matrix needed for the refinement of the Ritz vectors is computed and stored in the array B
DMDREFINE_E	'E': The unscaled eigenvectors of the Exact DMD are computed and returned in the array B
DMDREFINE_N	'N': No eigenvector refinement data is computed

ENUM_SVD_ALG

An enumeration selecting the SVD algorythm.

ID	Description
SVDALG_1	1: GESVD (the QR SVD algorithm)
SVDALG_2	2: GESDD (the Divide and Conquer algorithm)

ID	Description
SVDALG_3	3: GESVDQ (the preconditioned QR SVD; this and 4 are the most accurate options)
SVDALG_4	4: GEJSV (the preconditioned Jacobi SVD; this and 3 are the most accurate options)

DynamicModeDecompositionQR

Compute the Dynamic Mode Decomposition (DMD) for a pair of data snapshot matrices, using a QR factorization based compression of the data. Lapack function GEDMDQ. For the input matrices X and Y such that Y = A*X with an unaccessible matrix A, GEDMDQ computes a certain number of Ritz pairs of A using the standard Rayleigh-Ritz extraction from a subspace of range(X) that is determined using the leading left singular vectors of X. Optionally, GEDMDQ returns the residuals of the computed Ritz pairs, the information needed for a refinement of the Ritz vectors, or the eigenvectors of the Exact DMD.

The input M-by-N matrix F. The columns of F are the sequence of data snapshots from a single trajectory, taken at equidistant discrete times. It is assumed that the column norms of F are in the range of the normalized floating point numbers.

Computing for type matrix<double>

```
bool  matrix::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                    // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                    // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                    // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                    // specifies whether to explicitly
   ENUM_DMDQ_R           jobt,                    // specifies whether to return the
   ENUM_DMD_REFINE       jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG          whtsvd,                  // allows for a selection of the SV
   long                  nrnk,                    // determines the mode how to comput
   double                tol,                     // the tolerance for truncating sma
   vectorc&              eigen_values,            // vector of eigenvalues (Ritz value
   matrix&               Q,                       // Q factor from the QR factorizati
   matrix&               R,                       // R factor from the QR factorizati
   matrix&               left_vectors,            // matrix of left singular vectors
   matrix&               Z                        // matrix of Ritz vectors
   vector&               residuals,               // residuals for the Ritz pairs
   matrix&               B,                       // computed vectors
   matrix&               V,                       // eigenvectors of the matrix Raylei
   matrix&               S                        // computed vectors
   );
```

Computing for type matrix<float>

```
bool  matrixf::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                    // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                    // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                    // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                    // specifies whether to explicitly
   ENUM_DMDQ_R           jobt,                    // specifies whether to return the
   ENUM_DMD_REFINE       jobf,                    // specifies whether to store infor
   ENUM_SVD_ALG          whtsvd,                  // allows for a selection of the SV
   long                  nrnk,                    // determines the mode how to comput
   float                 tol,                     // the tolerance for truncating sma
   vectorcf&             eigen_values,          // vector of eigenvalues (Ritz value
   matrixf&              Q,                     // Q factor from the QR factorizatio
   matrixf&              R,                     // R factor from the QR factorizatio
   matrixf&              left_vectors,          // matrix of left singular vectors
   matrixf&              Z                      // matrix of Ritz vectors
   vectorf&              residuals,             // residuals for the Ritz pairs
   matrixf&              B,                     // computed vectors
   matrixf&              V,                     // eigenvectors of the matrix Raylei
   matrixf&              S                      // computed vectors
   );
```

Computing for type matrix<complex>

```
bool  matrixc::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                  // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                  // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                  // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                  // specifies whether to explicitly c
   ENUM_DMDQ_R           jobt,                  // specifies whether to return the u
   ENUM_DMD_REFINE       jobf,                  // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,                // allows for a selection of the SVD
   long                  nrnk,                  // determines the mode how to comput
   double                tol,                   // the tolerance for truncating smal
   vectorc&              eigen_values,          // vector of eigenvalues (Ritz value
   matrixc&              Q,                     // Q factor from the QR factorizatio
   matrixc&              R,                     // R factor from the QR factorizatio
   matrixc&              left_vectors,          // matrix of left singular vectors
   matrixc&              Z                      // matrix of Ritz vectors
   vector&               residuals,             // residuals for the Ritz pairs
   matrixc&              B,                     // computed vectors
   matrixc&              V,                     // eigenvectors of the matrix Raylei
   matrixc&              S                      // computed vectors
   );
```

Computing for type matrix<complexf>

```
bool  matrixcf::DynamicModeDecompositionQR(
   ENUM_DMD_SCALE        jobs,                  // determines whether the initial da
   ENUM_DMDQ_EIGV        jobz,                  // determines whether the eigenvecto
   ENUM_DMD_RESIDUALS    jobr,                  // determines whether to compute the
   ENUM_DMDQ_Q           jobq,                  // specifies whether to explicitly c
   ENUM_DMDQ_R           jobt,                  // specifies whether to return the u
   ENUM_DMD_REFINE       jobf,                  // specifies whether to store inform
   ENUM_SVD_ALG          whtsvd,                // allows for a selection of the SVD
   long                  nrnk,                  // determines the mode how to comput
   float                 tol,                   // the tolerance for truncating smal
   vectorcf&             eigen_values,          // vector of eigenvalues (Ritz value
   matrixcf&             Q,                     // Q factor from the QR factorizatio
   matrixcf&           R,                   // R factor from the QR factorizatio
   matrixcf&           left_vectors,        // matrix of left singular vectors
   matrixcf&           Z                    // matrix of Ritz vectors
   vectorf&            residuals,           // residuals for the Ritz pairs
   matrixcf&           B,                   // computed vectors
   matrixcf&           V,                   // eigenvectors of the matrix Rayle
   matrixcf&           S                    // computed vectors
   );
```

Parameters

jobs

[in] Value from the ENUM_DMD_SCALE enumeration which determines whether the initial data snapshots are scaled by a diagonal matrix. The data snapshots are the columns of F. The leading N-1 columns of F are denoted X and the trailing N-1 columns are denoted Y.

jobz

[in] Value from the ENUM_DMDQ_EIGV enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

jobr

[in] Value from the ENUM_DMD_RESIDUALS enumeration which determines whether to compute the residuals.

jobq

[in] Value from the ENUM_DMDQ_Q enumeration which specifies whether to explicitly compute and return the unitary matrix from the QR factorization of the data snapshot matrix.

jobt

[in] Value from the ENUM_DMDQ_R enumeration which specifies whether to return the upper triangular factor from the QR factorization of the data snapshot matrix.

jobf

[in] Value from the ENUM_DMD_REFINE enumeration which specifies whether to store information needed for post-processing (e.g. computing refined Ritz vectors).

whtsvd

[in] Value from the ENUM_SVD_ALG enumeration which allows for a selection of the SVD algorithm from the LAPACK library.

nrnk

[in] Value determines the mode how to compute the numerical rank, i.e. how to truncate small singular values of the input matrix X. If

NRNK = -1 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(1). This option is recommended.

NRNK = -2 :: i-th singular value sigma(i) is truncated if sigma(i) <= TOL*sigma(i-1). This option is included for R&D purposes. It requires highly accurate SVD, which may not be feasible.

The numerical rank can be enforced by using positive value of NRNK as follows: 0 < NRNK <= N-1 :: at most NRNK largest singular values will be used. If the number of the computed nonzero singular values is less than NRNK, then only those nonzero values will be used and the actually used dimension is less than NRNK.

tol

[in] The tolerance for truncating small singular values. 0 <= TOL < 1

eigen_values

[out] Vector of eigenvalues of size K. The leading K (K<N) entries of EIGS contain the computed eigenvalues (Ritz values).

Q

[out] Orthogonal matrix/factor of the QR factorization of the initial data snapshots matrix F.

R key

[out] N-by-N upper triangular factor from the QR factorization of the data snapshot matrix F.

left_vectors

[out] Matrix of left singular vectors. The leading K columns of X contain the leading K left singular vectors of the hold representations of the leading N-1 snapshots in the orthonormal basis computed in the QR factorization of F. To lift them to the space of the left singular vectors U(:,1:K) of the input data, pre-multiply with the Q factor from the initial QR factorization.

Z

[out] M-by-K matrix

If JOBZ =='V' then Z contains the Ritz vectors. Z(:,i) is an eigenvector of the i-th Ritz value; ||Z(:,i)||_2=1.

If JOBZ == 'F', then the Z(:,i)'s are given implicitly as Z*V, where Z contains orthonormal matrix (the product of Q from the initial QR factorization and the SVD/POD_basis returned by DynamicModeDecomposition in X) and the second factor (the eigenvectors of the Rayleigh quotient) is in the matrix V, as returned by DynamicModeDecomposition. That is, X(:,1:K)*V(:,i) is an eigenvector corresponding to EIGS(i). The columns of V(1:K,1:K) are the computed eigenvectors of the K-by-K Rayleigh quotient.

residuals

[out] Residuals for the K computed Ritz pairs.

B

[out] N-by-K matrix (K<N).

If JOBF =='R', B(1:N,1:K) contains A*U(:,1:K), and can be used for computing the refined vectors.

If JOBF == 'E', B(1:N,1;K) contains A*U(:,1:K)*W(1:K,1:K), which are the vectors from the Exact DMD, up to scaling by the inverse eigenvalues.

In both cases, the content of B can be lifted to the original dimension of the input data by pre-multiplying with the Q factor from the initial QR factorization. Here A denotes a compression of the underlying operator.

If JOBF =='N', then B is not referenced.

V

[out] K-by-K matrix (K<N). Contains the the K eigenvectors of the Rayleigh quotient. The Ritz vectors (returned in Z) are the product of Q from the initial QR factorization.

S

[out] K-by-K matrix (K<N). The array S(1:K,1:K) is used for the matrix Rayleigh quotient. This content is overwritten during the eigenvalue decomposition by GEEV.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The number of matrices rows (M) must not be less than the number of columns (N).

ENUM_DMD_SCALE

An enumeration that determines whether the initial data snapshots are scaled by a diagonal matrix.

ID	Description
DMDSCALE_S	'S': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that X*D has unit nonzero columns
DMDSCALE_C	'C': The snapshots are scaled as with the 'S' option.If it is found that an i-th column of X is zero vector and the corresponding i-th column of Y is non-zero, then the i-th column of Y is set to zero
DMDSCALE_Y	'Y': The data snapshots matrices X and Y are multiplied with a diagonal matrix D so that Y*D has unit nonzero columns
DMDSCALE_N	'N': No data scaling

ENUM_DMDQ_EIGV

An enumeration which determines whether the eigenvectors (Koopman modes) will be computed.

ID	Description
DMDQEIGV_V	'V': The eigenvectors (Koopman modes) will be computed
DMDQEIGV_F	'F': The eigenvectors (Koopman modes) will be returned in factored form as the product Z*V
DMDQEIGV_Q	'Q': The eigenvectors (Koopman modes) will be returned in factored form as the product Q*Z
DMDQEIGV_N	'N': The eigenvectors are not computed

ENUM_DMD_RESIDUALS

An enumeration which determines whether to compute the residuals.

ID	Description
DMDRESIDUALS_R	'R': The residuals for the computed eigenpairs will be computed
DMDRESIDUALS_N	'N': The residuals are not computed

ENUM_DMDQ_Q

An enumeration which specifies whether to explicitly compute and return the unitary matrix from the QR factorization.

ID	Description
DMDQQ_Q	'Q': The matrix Q of the QR factorization of the data snapshot matrix is computed
DMDQQ_N	'N': The matrix Q is not explicitly computed

ENUM_DMDQ_R

An enumeration whichspecifies whether to return the upper triangular factor from the QR factorization.

ID	Description
DMDQR_R	'R': The matrix R of the QR factorization of the data snapshot matrix is computed
DMDQR_N	'N': The matrix R is not explicitly computed

ENUM_DMD_REFINE

An enumeration which specifies whether to store information needed for post-processing.

ID	Description
DMDREFINE_R	'R': The matrix needed for the refinement of the Ritz vectors is computed and stored in the array B
DMDREFINE_E	'E': The unscaled eigenvectors of the Exact DMD are computed and returned in the array B

ID	Description
DMDREFINE_N	'N': No eigenvector refinement data is computed

ENUM_SVD_ALG

An enumeration selecting the SVD algorythm.

ID	Description
SVDALG_1	1: GESVD (the QR SVD algorithm)
SVDALG_2	2: GESDD (the Divide and Conquer algorithm)
SVDALG_3	3: GESVDQ (the preconditioned QR SVD; this and 4 are the most accurate options)
SVDALG_4	4: GEJSV (the preconditioned Jacobi SVD; this and 3 are the most accurate options)

Singular Spectrum Analysis

The section contains functions for decomposing a matrix into three components: orthogonal matrices and a diagonal matrix of singular values. SVD is applied to solve various linear algebra problems such as data dimensionality reduction, image compression, solving systems of equations, as well as data analysis and optimization. The main functions allow calculating singular values and vectors, reconstruct matrices, as well as approximate them with reduced rank accuracy.

Function	Action
SingularSpectrumAnalysisSpectrum	A method function for calculating the relative contributions of spectral components based on their eigenvalues.
SingularSpectrumAnalysisForecast	A method function for calculating reconstructed and predicted data using spectral components of the input time series.
SingularSpectrumAnalysisReconstructComponents	A method function for calculating reconstructed components of the input time series and their contributions.
SingularSpectrumAnalysisReconstructSeries	A method function for calculating the reconstructed time series using the first component_count components.

SingularSpectrumAnalysisSpectrum

A method function for calculating the relative contributions of spectral components based on their eigenvalues.

Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisSpectrum(
   ulong    window_length,       // window size for constructing the trajectory matrix
   vector& spectrum              // vector of component contributions to the input serie
   );
```

Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisSpectrum(
   ulong    window_length,       // window size for constructing the trajectory matrix
   vectorc& spectrum             // vector of component contributions to the input serie
   );
```

Parameters

window_length

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

spectrum

[out] Vector of component contributions to the input series - eigenvalues of the covariance matrix of the input time series.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

SingularSpectrumAnalysisForecast

A method function for calculating reconstructed and predicted data using spectral components of the input time series.

Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisForecast(
   ulong     window_length,          // window size for constructing the trajectory matr
   ulong     component_count,        // number of components used for forecasting
   ulong     forecast_horizon,       // number of points to forecast
   vector&   forecast                // vector consisting of reconstructed and predicte
   );
```

Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisForecast(
   ulong     window_length,          // window size for constructing the trajectory matr
   ulong     component_count,        // number of components used for forecasting
   ulong     forecast_horizon,       // number of points to forecast
   vectorc&  forecast                // vector consisting of reconstructed and predicte
   );
```

Parameters

window_length

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

component_count

[in] Number of components used for forecasting.

forecast_horizon

[in] Number of points to forecast.

forecast

[out] Combining points reconstructed by *component_count* plus *forecast_horizon* points predicted using the first *component_count* components. Thus, the *forecast* vector has the size of (T+forecast_horizon), where T is the input series length.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

SingularSpectrumAnalysisReconstructComponents

A method function for calculating reconstructed components of the input time series and their contributions.

Calculations for vector<double> and matrix<double> types

```
bool  vector::SingularSpectrumAnalysisReconstructComponents(
   ulong     window_length,       // window size for constructing the trajectory matrix
   matrix&   components,          // matrix of reconstructed components
   vector&   contributions        // vector of component contributions to the input ser
   );
```

Calculations for vector<complex> and matrix<complex> types

```
bool  vector::SingularSpectrumAnalysisReconstructComponents(
   ulong     window_length,       // window size for constructing the trajectory matrix
   matrixc&  components,          // matrix of reconstructed components
   vectorc&  contributions        // vector of component contributions to the input ser
   );
```

Parameters

`window_length`

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

`components`

[out] A matrix of reconstructed components, where each column describes a separate component.

`contributions`

[out] Vector of component contributions to the input series (eigenvalues of the covariance matrix of the input time series).

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

SingularSpectrumAnalysisReconstructSeries

A method function for calculating the reconstructed time series using the first *component_count* components.

Calculations for vector<double> type

```
bool  vector::SingularSpectrumAnalysisReconstructSeries(
   ulong     window_length,       // window size for constructing the trajectory matrix
   ulong     component_count,     // number of components used for reconstruction
   vector&   reconstructed        // reconstructed time series
   );
```

Calculations for vector<complex> type

```
bool  vector::SingularSpectrumAnalysisReconstructSeries(
   ulong     window_length,       // window size for constructing the trajectory matrix
   ulong     component_count,     // number of components used for reconstruction
   vectorc&  reconstructed        // reconstructed time series
   );
```

Parameters

window_length

[in] Window size for constructing the trajectory matrix, the number of components the input time series should be decomposed into.

component_count

[out] Number of components used for time series reconstruction.

reconstructed

[out] Vector containing the reconstructed output series.

Return Value

The function returns 'true' on success or 'false' if an error occurs.

Note

The *window_length* parameter value should be less than the size of the input time series. To construct a full-fledged trajectory matrix, the optimal size is considered to be approximately equal to half the size of the input time series.

Reductions

Functions designed to transform matrices into special forms using orthogonal transformations. These methods are widely used in numerical computations, such as spectral analysis, eigenvalue problems, and singular value decomposition (SVD) of matrices.

These functions enable efficient matrix transformations essential for solving linear algebra problems, including computing Singular Value Decomposition (SVD) and finding eigenvalues of symmetric matrices. The use of LAPACK functions ensures high performance and computational accuracy.

Function	Action
ReduceToBidiagonal	Reduces a general real or complex m-by-n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: Q**T * A * P = B. If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal. Lapack function GEBRD.
ReflectBidiagonalToQP	Generates orthogonal matrices Q and P**T (or P**H for complex types) determined by ReduceToBidiagonal method when reducing a real or complex matrix A to bidiagonal form: A = Q * B * P**T. Q and P**T are defined as products of elementary reflectors H(i) or G(i) respectively. Lapack functions ORGBR, UNGBR.
ReduceSymmetricToTridiagonal	Reduces a real symmetric or complex Hermitian matrix A to tridiagonal form B by an orthogonal similarity transformation: Q**T * A * Q = B. Lapack functions SYTRD, HETRD.
ReflectTridiagonalToQ	Generates orthogonal matrix Q, which is defined as the product of n-1 elementary reflectors of order n, as returned by ReduceSymmetricToTridiagonal

ReduceToBidiagonal

Reduces a general real or complex m-by-n matrix A to upper or lower bidiagonal form B by an orthogonal transformation: Q**T * A * P = B. If m >= n, B is upper bidiagonal; if m < n, B is lower bidiagonal. Lapack function GEBRD.

Computing for type matrix<double>

```
bool  matrix::ReduceToBidiagonal(
   matrix&         B,             // bidiagonal matrix
   matrix&         reflect_qp,    // q and p-reflectors
   vector&         tau_q,         // scalar factors of the elementary reflectors Q
   vector&         tau_p          // scalar factors of the elementary reflectors P
   );
```

Computing for type matrix<float>

```
bool  matrix::ReduceToBidiagonal(
   matrixf&        B,             // bidiagonal matrix
   matrixf&        reflect_qp,    // q and p-reflectors
   vectorf&        tau_q,         // scalar factors of the elementary reflectors Q
   vectorf&        tau_p          // scalar factors of the elementary reflectors P
   );
```

Computing for type matrix<complex>

```
bool  matrix::ReduceToBidiagonal(
   matrixc&        B,             // bidiagonal matrix
   matrixc&        reflect_qp,    // q and p-reflectors
   vectorc&        tau_q,         // scalar factors of the elementary reflectors Q
   vectorc&        tau_p          // scalar factors of the elementary reflectors P
   );
```

Computing for type matrix<complexf>

```
bool  matrix::ReduceToBidiagonal(
   matrixcf&       B,             // bidiagonal matrix
   matrixcf&       reflect_qp,    // q and p-reflectors
   vectorcf&       tau_q,         // scalar factors of the elementary reflectors Q
   vectorcf&       tau_p          // scalar factors of the elementary reflectors P
   );
```

Parameters

B

[out] Upper or lower bidiagonal matrix.

reflect_qp

[out] Transformed matrix A. If m >= n, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the vector tau_q, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the vector tau_p, represent the orthogonal matrix P as a product of

elementary reflectors; if m < n, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the vector tau_q, represent the orthogonal matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the vector tau_p, represent the orthogonal matrix P as a product of elementary reflectors.

tau_q

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

tau_p

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix P.

Return Value

Return true if successful, otherwise false in case of an error.

Note

The matrices Q and P are represented as products of elementary reflectors:

If m >= n,

Q = H(1) H(2) . . . H(n) and P = G(1) G(2) . . . G(n-1)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v**T and G(i) = I - taup * u * u**T

where tauq and taup are scalars, and v and u are vectors;

v(1:i-1) = 0, v(i) = 1, and v(i+1:m) is stored on exit in A(i+1:m,i);

u(1:i) = 0, u(i+1) = 1, and u(i+2:n) is stored on exit in A(i,i+2:n);

tauq is stored in *tau_q*(i) and taup in *tau_p*(i).

If m < n,

Q = H(1) H(2) . . . H(m-1) and P = G(1) G(2) . . . G(m)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v**T and G(i) = I - taup * u * u**T

where tauq and taup are scalars, and v and u are vectors;

v(1:i) = 0, v(i+1) = 1, and v(i+2:m) is stored on exit in A(i+2:m,i);

u(1:i-1) = 0, u(i) = 1, and u(i+1:n) is stored on exit in A(i,i+1:n);

tauq is stored in *tau_q*(i) and taup in *tau_p*(i).

The contents of A on exit (output matrix *reflect_qp*) are illustrated by the following examples:

```
     m = 6 and n = 5 (m > n):           m = 5 and n = 6 (m < n):
       (  d   e   u1  u1  u1 )            (  d   u1  u1  u1  u1  u1 )
       (  v1  d   e   u2  u2 )            (  e   d   u2  u2  u2  u2 )
       (  v1  v2  d   e   u3 )            (  v1  e   d   u3  u3  u3 )
       (  v1  v2  v3  d   e  )            (  v1  v2  e   d   u4  u4 )
       (  v1  v2  v3  v4  d  )            (  v1  v2  v3  e   d   u5 )
       (  v1  v2  v3  v4  v5 )
```

where d and e denote diagonal and off-diagonal elements of B, vi denotes an element of the vector defining H(i), and ui an element of the vector defining G(i).

Matrices Q and P can be produced with ReflectBidiagonalToQP method.

ReflectBidiagonalToQP

Generates orthogonal matrices Q and P**T (or P**H for complex types) determined by ReduceToBidiagonal method when reducing a real or complex matrix A to bidiagonal form: A = Q * B * P**T. Q and P**T are defined as products of elementary reflectors H(i) or G(i) respectively. Lapack functions ORGBR, UNGBR.

As input is used transformed matrix reflect_qp with the same sizes m-by-n as in original matrix A.

Computing for type matrix<double>

```
bool  matrix::ReflectBidiagonalToQP(
   vector&          tau_q,          // scalar factors of the elementary reflectors Q
   vector&          tau_p,          // scalar factors of the elementary reflectors P
   matrix&          Q,              // matrix Q
   matrix&          PT              // transposed matrix P
   );
```

Computing for type matrix<float>

```
bool  matrix::ReflectBidiagonalToQP(
   vectorf&         tau_q,          // scalar factors of the elementary reflectors Q
   vectorf&         tau_p,          // scalar factors of the elementary reflectors P
   matrixf&         Q,              // matrix Q
   matrixf&         PT              // transposed matrix P
   );
```

Computing for type matrix<complex>

```
bool  matrix::ReflectBidiagonalToQP(
   vectorc&         tau_q,          // scalar factors of the elementary reflectors Q
   vectorc&         tau_p,          // scalar factors of the elementary reflectors P
   matrixc&         Q,              // matrix Q
   matrixc&         PH              // hermitian conjugated matrix P
   );
```

Computing for type matrix<complexf>

```
bool  matrix::ReflectBidiagonalToQP(
   vectorcf&        tau_q,          // scalar factors of the elementary reflectors Q
   vectorcf&        tau_p,          // scalar factors of the elementary reflectors P
   matrixcf&        Q,              // matrix Q
   matrixcf&        PH              // hermitian conjugated matrix P
   );
```

Parameters

tau_q

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

tau_p

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix P.

Q

[out] Orthogonal matrix Q.

PT (PH)

[out] Transposed (or hermitian conjugated) matrix P.

Return Value

Return true if successful, otherwise false in case of an error.

Note

If m >= n, matrix Q is of m-by-n sizes, matrix PT is of n-by-n sizes.

If m < n, matrix Q is of m-by-m sizes, matrix PT is of m-by-n sizes.

ReduceSymmetricToTridiagonal

Reduces a real symmetric or complex Hermitian matrix A to trdiagonal form B by an orthogonal similarity transformation: Q**T * A * Q = B. Lapack functions SYTRD, HETRD.

Computing for type matrix<double>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrix&          B,              // tridiagonal matrix
   matrix&          reflect_q,      // q-reflectors
   vector&          tau_q           // scalar factors of the elementary reflectors Q
   );
```

Computing for type matrix<float>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixf&         B,              // tridiagonal matrix
   matrixf&         reflect_q,      // q-reflectors
   vectorf&         tau_q           // scalar factors of the elementary reflectors Q
   );
```

Computing for type matrix<complex>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixc&         B,              // tridiagonal matrix
   matrixc&         reflect_q,      // q-reflectors
   vectorc&         tau_q           // scalar factors of the elementary reflectors Q
   );
```

Computing for type matrix<complexf>

```
bool  matrix::ReduceSymmetricToTridiagonal(
   matrixcf&        B,              // tridiagonal matrix
   matrixcf&        reflect_q,      // q and p-reflectors
   vectorcf&        tau_q           // scalar factors of the elementary reflectors Q
   );
```

Parameters

B

[out] Symmetric (or Hermitian) tridiagonal matrix.

reflect_q

[out] Upper or lower triangular matrix, it depends on the input matrix A. In upper case the diagonal and first superdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix B, and the elements above the first superdiagonal, with the array *tau_q*, represent the orthogonal matrix Q as a product of elementary reflectors; in lower case the diagonal and first subdiagonal of A are overwritten by the corresponding elements of the tridiagonal matrix B, and the elements below the first subdiagonal, with the array *tau_q*, represent the orthogonal matrix Q as a product of elementary reflectors.

tau_q

[out] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

Return Value

Return true if successful, otherwise false in case of an error.

Note

In upper case the matrix Q is represented as a product of elementary reflectors

Q = H(n-1) . . . H(2) H(1)

Each H(i) has the form

H(i) = I - tau * v * v**T

where tau is a scalar, and v is a vector with v(i+1:n) = 0 and v(i) = 1; v(1:i-1) is stored on exit in A(1:i-1,i+1), and tau in `tau_q`(i).

In lower case the matrix Q is represented as a product of elementary reflectors

Q = H(1) H(2) . . . H(n-1)

Each H(i) has the form

H(i) = I - tau * v * v**T

where tau is a scalar, and v is a vector with v(1:i) = 0 and v(i+1) = 1; v(i+2:n) is stored on exit in A(i+2:n,i), and tau in `tau_q`(i). */

The contents of A on exit (output matrix `reflect_q`) are illustrated by the following examples with n = 5:

```
     upper case:                        lower case:

     (  d   e   v2  v3  v4 )            (  d                   )
     (      d   e   v3  v4 )            (  e   d               )
     (          d   e   v4 )            (  v1  e   d           )
     (              d   e  )            (  v1  v2  e   d       )
     (                  d  )            (  v1  v2  v3  e   d   )
```

where d and e denote diagonal and off-diagonal elements of B and vi denotes an element of the vector defining H(i).

Matrix Q can be produced with ReflectTridiagonalToQ method.

ReflectTridiagonalToQ

Generates orthogonal matrix Q which is defined as the product of n-1 elementary reflectors of order n, as returned by ReduceSymmetricToTridiagonal:

in upper case Q = H(n-1) . . . H(2) H(1),

in lower case Q = H(1) H(2) . . . H(n-1).

Lapack functions ORGTR, UNGTR.

As input is used transformed matrix reflect_q with the same sizes n-by-n as in original matrix A.

Computing for type matrix<double>

```
bool  matrix::ReflectTridiagonalToQ(
   vector&          tau_q,          // scalar factors of the elementary reflectors Q
   matrix&          Q               // matrix Q
   );
```

Computing for type matrix<float>

```
bool  matrix::ReflectTridiagonalToQ(
   vectorf&         tau_q,          // scalar factors of the elementary reflectors Q
   matrixf&         Q               // matrix Q
   );
```

Computing for type matrix<complex>

```
bool  matrix::ReflectTridiagonalToQ(
   vectorc&         tau_q,          // scalar factors of the elementary reflectors Q
   matrixc&         Q               // matrix Q
   );
```

Computing for type matrix<complexf>

```
bool  matrix::ReflectTridiagonalToQ(
   vectorcf&        tau_q,          // scalar factors of the elementary reflectors Q
   matrixcf&        Q               // matrix Q
   );
```

Parameters

tau_q

[in] Vector of the scalar factors of the elementary reflectors which represent the orthogonal matrix Q.

Q

[out] Orthogonal matrix Q.

Return Value

Return true if successful, otherwise false in case of an error.

Conversion Functions

This is a group of functions that provide conversion of data from one format into another.

The NormalizeDouble() function must be specially noted as it provides the necessary accuracy of the price presentation. In trading operations, no unnormalized prices may be used if their accuracy even a digit exceeds that required by the trade server.

Function	Action
CharToString	Converting a symbol code into a one-character string
DoubleToString	Converting a numeric value to a text line with a specified accuracy
EnumToString	Converting an enumeration value of any type to string
NormalizeDouble	Rounding of a floating point number to a specified accuracy
StringToDouble	Converting a string containing a symbol representation of number into number of double type
StringToInteger	Converting a string containing a symbol representation of number into number of long type
StringToTime	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type
TimeToString	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format
IntegerToString	Converting int into a string of preset length
ShortToString	Converting symbol code (unicode) into one-symbol string
ShortArrayToString	Copying array part into a string
StringToShortArray	Symbol-wise copying a string to a selected part of array of ushort type
CharArrayToString	Converting symbol code (ansi) into one-symbol array
StringToCharArray	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type
CharArrayToStruct	Copy uchar type array to POD structure
StructToCharArray	Copy POD structure to uchar type array
ColorToARGB	Converting color type to uint type to receive ARGB representation of the color.
ColorToString	Converting color value into string as "R,G,B"
StringToColor	Converting "R,G,B" string or string with color name into color type value
StringFormat	Converting number into string according to preset format

See also

[Use of a Codepage]

CharToString

Converting a symbol code into a one-character string.

```
string  CharToString(
   uchar  char_code       // numeric code of symbol
   );
```

Parameters

char_code

[in] Code of ANSI symbol.

Return Value

String with a ANSI symbol.

Example:

```
string ExtStrArray[224];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- in the loop from the symbol code 32 (space) to 255(UCHAR_MAX),
//--- fill the array with symbol codes converted to a string in accordance with the cu
   for(int i=32; i<=UCHAR_MAX; i++)
      ExtStrArray[i-32]=CharToString((uchar)i);

//--- print the header and the symbol table in the journal
   Print("Table of symbols:");
   ArrayPrint(ExtStrArray,_Digits," | ");
   /*
   result:
   Table of symbols:
   [  0] " " | "!" | """ | "#" | "$" | "%" | "&" | "'" | "(" | ")" | "*" | "+" | ","
   [ 25] "9" | ":" | ";" | "<" | "=" | ">" | "?" | "@" | "A" | "B" | "C" | "D" | "E"
   [ 50] "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "[" | "\" | "]" | "^"
   [ 75] "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w"
   [100] "„" | "…" | "†" | "‡" | "€" | "‰" | "Љ" | "‹" | "Њ" | "Ќ" | "Ћ" | "Џ" | "ђ"
   [125] "ќ" | "ћ" | "џ" | " " | "Ў" | "ў" | "Ј" | "¤" | "Ґ" | "¦" | "§" | "Ё" | "©"
   [150] "¶" | "·" | "ё" | "№" | "є" | "»" | "ј" | "Ѕ" | "ѕ" | "ї" | "А" | "Б" | "В"
   [175] "П" | "Р" | "С" | "Т" | "У" | "Ф" | "Х" | "Ц" | "Ч" | "Ш" | "Щ" | "Ъ" | "Ы"
   [200] "и" | "й" | "к" | "л" | "м" | "н" | "о" | "п" | "р" | "с" | "т" | "у" | "ф"
   */
  }
```

See also

StringToCharArray, ShortToString, StringGetCharacter

CharArrayToString

It copies and converts part of array of uchar type into a returned string.

```
string  CharArrayToString(
   uchar  array[],             // array
   int    start=0,             // starting position in the array
   int    count=-1             // number of symbols
   uint    codepage=CP_ACP     // code page
   );
```

Parameters

array[]

[in] Array of uchar type.

start=0

[in] Position from which copying starts. by default 0 is used.

count=-1

[in] Number of array elements for copying. Defines the length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.

codepage=CP_ACP

[in] The value of the code page. There is a number of built-in constants for the most used code pages.

Return Value

String.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create an array with symbol codes
   uchar array[]= { 84, 104, 105, 115,  32, 105, 115,  32,  97,  32,
                   116, 101, 115, 116,  32, 111, 102,  32, 116, 104,
                   101,  32,  67, 104,  97, 114,  65, 114, 114,  97,
                   121,  84, 111,  83, 116, 114, 105, 110, 103,  40,
                    41,  32, 102, 117, 110,  99, 116, 105, 111, 110
                  };
//--- print an array of codes converted into a string in the journal
   Print(CharArrayToString(array));

//--- create an array with symbol codes and terminal zero
   uchar array_z[]= { 84, 104, 105, 115,  32, 105, 115,  32,  97,  32,
                      116, 101, 115, 116,   0,  32, 111, 102,  32, 116,
                      104, 101,  32,  67, 104,  97, 114,  65, 114, 114,
                       97, 121,  84, 111,  83, 116, 114, 105, 110, 103,
                       40,  41,  32, 102, 117, 110,  99, 116, 105, 111, 110
                    };
//--- print the second array of codes with terminal zero in the journal
   PrintFormat("%s ...", CharArrayToString(array_z));

   /*
   result:
   This is a test of the CharArrayToString() function
   This is a test ...
   */
  }
```

See also

StringToCharArray, ShortArrayToString, Use of a Codepage

CharArrayToStruct

Copy uchar type array to POD structure.

```
bool  CharArrayToStruct(
   void&          struct_object,    // structure
   const uchar&   char_array[],     // array
   uint           start_pos=0       // starting position in the array
   );
```

Parameters

struct_object

[in] Reference to any type of POD structure (containing only simple data types).

char_array[]

[in] uchar type array.

start_pos=0

[in] Position in the array, data copying starts from.

Return Value

Returns true if successful, otherwise false.

Example:

```
#define DATA_TOTAL 4

MqlRates ExtDataRates[DATA_TOTAL];
uchar    ExtCharArray[];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- copy the data of the last four bars into the uchar array
   ResetLastError();
   for(int i=0; i<DATA_TOTAL; i++)
     {
      if(!CopyRatesToCharArray(i, ExtCharArray))
         return;
     }

//--- copy all available data into the MqlRates structure array in a loop by the amour
   for(int i=0; i<DATA_TOTAL; i++)
     {
      ResetLastError();
      if(!CharArrayToStruct(ExtDataRates[i], ExtCharArray, sizeof(MqlRates)*i))
     {
      Print("CharArrayToStruct() failed. Error code: ", GetLastError());
      return;
     }
   //--- upon successful copying of data from the uchar array to the MqlRates struc
   //--- print the data, copied into the structure, to the journal
   MqlRatesPrint(ExtDataRates[i]);
  }

/*
result:
MqlRates data:
 Time = 2024.02.21 09:00;
 Open = 1.08116;
 High = 1.08141;
 Low = 1.08062;
 Close = 1.08124;
 Tick volume = 5344;
 Spread = 1;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 08:00;
 Open = 1.08149;
 High = 1.08153;
 Low = 1.08073;
 Close = 1.08114;
 Tick volume = 3607;
 Spread = 2;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 07:00;
 Open = 1.08143;
 High = 1.08165;
 Low = 1.08122;
 Close = 1.08150;
 Tick volume = 2083;
 Spread = 0;
 Real volume = 0

MqlRates data:
 Time = 2024.02.21 06:00;
 Open = 1.08178;
 High = 1.08190;
 Low = 1.08132;
 Close = 1.08142;
 Tick volume = 1733;
 Spread = 0;
```

```
    Real volume = 0
   */
  }
//+------------------------------------------------------------------+
//| Copy bar data to the uchar array                                 |
//| by the specified index via the MqlRates structure                |
//+------------------------------------------------------------------+
bool CopyRatesToCharArray(const int index, uchar &data_array[])
  {
   MqlRates rates[1];
   uint     start=sizeof(MqlRates);

   ResetLastError();
   if(CopyRates(Symbol(), PERIOD_CURRENT, index, 1, rates)!=1)
     {
      Print("CopyRates() failed. Error code: ", GetLastError());
      return(false);
     }
   if(!StructToCharArray(rates[0], data_array, start*index))
     {
      Print("StructToCharArray() failed. Error code: ", GetLastError());
      return(false);
     }

   return(true);
  }
//+------------------------------------------------------------------+
//| Print the fields of the MqlRates structure in the journal        |
//+------------------------------------------------------------------+
void MqlRatesPrint(MqlRates &rates)
  {
//--- create the output string
   string text=StringFormat(" Time = %s;\n"
                            " Open = %.*f;\n"
                            " High = %.*f;\n"
                            " Low = %.*f;\n"
                            " Close = %.*f;\n"
                            " Tick volume = %I64u;\n"
                            " Spread = %d;\n"
                            " Real volume = %I64u",
                            TimeToString(rates.time),
                            _Digits, rates.open,
                            _Digits, rates.high,
                            _Digits, rates.low,
                            _Digits, rates.close,
                            rates.tick_volume,
                            rates.spread,
                            rates.real_volume);
//--- print the header and the output string in the journal
```

```
   Print("MqlRates data:\n", text, "\n");
  }
```

See also

StringToCharArray, ShortArrayToString, StructToCharArray, Use of a Codepage, FileReadStruct, Unions (union), MathSwap

StructToCharArray

Copy POD structure to uchar type array.

```
bool  StructToCharArray(
   const void&  struct_object,     // structure
   uchar&       char_array[],      // array
   uint         start_pos=0        // starting position in the array
   );
```

Parameters

struct_object

[in] Reference to any type of POD structure (containing only simple data types).

char_array[]

[in] uchar type array.

start_pos=0

[in] Position in the array, starting from which the copied data are added.

Return Value

Returns true if successful, otherwise false.

Note

When copying, the dynamic array automatically expands (ArrayResize) if there is not enough space. If the array cannot be expanded up to the required value, the function returns an error.

Example:

```
uchar     ExtCharArray[];
MqlRates ExtRates[1];

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- copy the data of one current bar into the MqlRates structure
   if(CopyRates(Symbol(), PERIOD_CURRENT, 0, 1, ExtRates)!=1)
     {
      Print("CopyRates() failed. Error code: ", GetLastError());
      return;
     }

//--- print the data received in the MqlRates structure in the journal
   Print("Data in the structure immediately after receiving it:");
   MqlRatesPrint(ExtRates[0]);
```

```
//--- copy the structure to the uchar type array
   ResetLastError();
   if(!StructToCharArray(ExtRates[0], ExtCharArray))
     {
      Print("StructToCharArray() failed. Error code: ", GetLastError());
      return;
     }

//--- clear the structure
   ZeroMemory(ExtRates[0]);
//--- print the data in the structure after clearing
   Print("\nData in the structure after ZeroMemory():");
   MqlRatesPrint(ExtRates[0]);

//--- now copy the data from the uchar array to the MqlRates structure
   if(!CharArrayToStruct(ExtRates[0], ExtCharArray))
     {
      Print("CharArrayToStruct() failed. Error code: ", GetLastError());
      return;
     }
//--- print the data in the MqlRates structure after copying it from the uchar array
   Print("\nData in the MqlRates structure after copying it from a uchar array:");
   MqlRatesPrint(ExtRates[0]);

   /*
   result:

   Data in the structure immediately after receiving it:
   MqlRates data:
    Time = 2024.02.21 07:00;
    Open = 1.08143;
    High = 1.08158;
    Low = 1.08122;
    Close = 1.08137;
    Tick volume = 1341;
    Spread = 4;
    Real volume = 0

   Data in the structure after ZeroMemory():
   MqlRates data:
    Time = 0;
    Open = 0.00000;
    High = 0.00000;
    Low = 0.00000;
    Close = 0.00000;
    Tick volume = 0;
    Spread = 0;
    Real volume = 0
   Data in the MqlRates structure after copying it from a uchar array:
   MqlRates data:
    Time = 2024.02.21 07:00;
    Open = 1.08143;
    High = 1.08158;
    Low = 1.08122;
    Close = 1.08137;
    Tick volume = 1341;
    Spread = 4;
    Real volume = 0
   */
  }
//+------------------------------------------------------------------+
//| Print the fields of the MqlRates structure in the journal        |
//+------------------------------------------------------------------+
void MqlRatesPrint(MqlRates &rates)
  {
//--- create the output string
   string text=StringFormat(" Time = %s;\n"
                            " Open = %.*f;\n"
                            " High = %.*f;\n"
                            " Low = %.*f;\n"
                            " Close = %.*f;\n"
                            " Tick volume = %I64u;\n"
                            " Spread = %d;\n"
                            " Real volume = %I64u",
                            TimeToString(rates.time),
                            _Digits, rates.open,
                            _Digits, rates.high,
                            _Digits, rates.low,
                            _Digits, rates.close,
                            rates.tick_volume,
                            rates.spread,
                            rates.real_volume);
//--- print the header and the output string in the journal
   Print("MqlRates data:\n", text);
  }
```

See also

StringToCharArray, ShortArrayToString,CharArrayToStruct, Use of a Codepage, FileWriteStruct, Unions (union), MathSwap

ColorToARGB

The function converts color type into uint type to get ARGB representation of the color. ARGB color format is used to generate a graphical resource, text display, as well as for CCanvas standard library class.

```
uint  ColorToARGB(
   color   clr,          // converted color in color format
   uchar   alpha=255     // alpha channel managing color transparency
   );
```

Parameters

clr

[in] Color value in color type variable.

alpha

[in] The value of the alpha channel used to receive the color in ARGB format. The value may be set from 0 (a color of a foreground pixel does not change the display of an underlying one) up to 255 (a color of an underlying pixel is completely replaced by the foreground pixel's one). Color transparency in percentage terms is calculated as (1-alpha/255)*100%. In other words, the lesser value of the alpha channel leads to more transparent color.

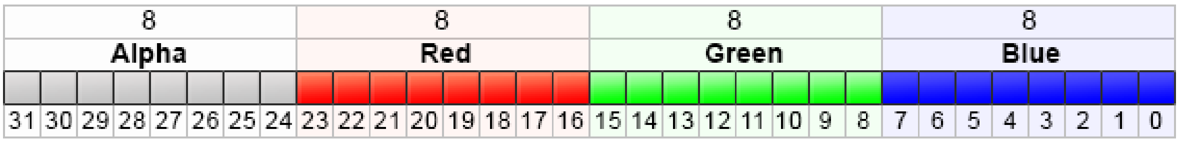
Return Value

Presenting the color in ARGB format where Alfa, Red, Green, Blue (alpha channel, red, green, blue) values are set in series in four uint type bytes.

Note

RGB is a basic and commonly used format for pixel color description on a screen in computer graphics. Names of basic colors are used to set red, green and blue color components. Each component is described by one byte specifying the color saturation in the range of 0 to 255 (0x00 to 0XFF in hexadecimal format). Since the white color contains all colors, it is described as 0xFFFFFF, that is, each one of three components is presented by the maximum value of 0xFF.

However, some tasks require to specify the color transparency to describe the look of an image in case it is covered by the color with some degree of transparency. The concept of alpha channel is introduced for such cases. It is implemented as an additional component of RGB format. ARGB format structure is shown below.



8	8	8	8
Alpha	**Red**	**Green**	**Blue**
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

ARGB values are typically expressed using hexadecimal format with each pair of digits representing the values of Alpha, Red, Green and Blue channels, respectively. For example, 80FFFF00 color represents 50.2% opaque yellow. Initially, 0x80 sets 50.2% alpha value, as it is 50.2% of 0xFF value. Then, the first FF pair defines the highest value of the red component; the next FF pair is like the previous but for the green component; the final 00 pair represents the lowest value the blue component can have (absence of blue). Combination of green and red colors yields yellow one. If the alpha channel is not used, the entry can be reduced down to 6 RRGGBB digits, this is why the alpha channel values are stored in the top bits of uint integer type.

Depending on the context, hexadecimal digits can be written with '0x' or '#' prefix, for example, 80FFFF00, 0x80FFFF00 or #80FFFF00.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set transparency
   uchar alpha=0x55;  // 0x55 means 55/255=21.6 % of transparency
   //--- derive conversion to ARGB for clrBlue color
   PrintFormat("0x%.8X - clrBlue",clrBlue);
   PrintFormat("0x%.8X - clrBlue ARGB with alpha=0x55 (transparency 21.6%%)",ColorToA
   //--- derive conversion to ARGB for clrGreen color
   PrintFormat("0x%.8X - clrGreen",clrGreen);
   PrintFormat("0x%.8X - clrGreen ARGB with alpha=0x55 (transparency 21.6%%)",ColorTo
   //--- derive conversion to ARGB for clrRed color
   PrintFormat("0x%.8X - clrRed",clrRed);
   PrintFormat("0x%.8X - clrRed ARGB with alpha=0x55 (transparency 21.6%%)",ColorToAR
  }
```

See also

Resources, ResourceCreate(), TextOut(), color type, char, short, int and long types

ColorToString

It converts color value into string of "R,G,B" form.

```
string  ColorToString(
   color  color_value,      // color value
   bool   color_name        // show color name or not
   );
```

Parameters

color_value

[in] Color value in color type variable.

color_name

[in] Return color name if it is identical to one of predefined color constants.

Return Value

String presentation of color as "R,G,B", where R, G and B are decimal constants from 0 to 255 converted into a string. If the color_name=true parameter is set, it will try to convert color value into color name.

Example:

```
   string clr=ColorToString(C'0,255,0'); // green color
   Print(clr);

   clr=ColorToString(C'0,255,0',true);   // get color constant
   Print(clr);
```

See also

StringToColor, ColorToARGB

DoubleToString

Converting numeric value into text string.

```
string  DoubleToString(
   double  value,      // number
   int     digits=8    // number of digits after decimal point
   );
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format. If the *digits* value is in the range between 0 and 16, a string presentation of a number with the specified number of digits after the point will be obtained. If the *digits* value is in the range between -1 and -16, a string representation of a number in the scientific format with the specified number of digits after the decimal point will be obtained. In all other cases the string value will contain 8 digits after the decimal point.

Return Value

String containing a symbol representation of a number with the specified accuracy.

Example:

```
   Print("DoubleToString(120.0 + M_PI) : ",DoubleToString(120.0+M_PI));
   Print("DoubleToString(120.0 + M_PI,16) : ",DoubleToString(120.0+M_PI,16));
   Print("DoubleToString(120.0 + M_PI,-16) : ",DoubleToString(120.0+M_PI,-16));
   Print("DoubleToString(120.0 + M_PI,-1) : ",DoubleToString(120.0+M_PI,-1));
   Print("DoubleToString(120.0 + M_PI,-20) : ",DoubleToString(120.0+M_PI,-20));
```

See also

NormalizeDouble, StringToDouble

EnumToString

Converting an enumeration value of any type to a text form.

```
string  EnumToString(
   any_enum  value      // any type enumeration value
   );
```

Parameters

value

[in] Any type enumeration value.

Return Value

A string with a text representation of the enumeration. To get the error message call the GetLastError() function.

Note

The function can set the following error values in the _LastError variable:

- ERR_INTERNAL_ERROR – error of the execution environment
- ERR_NOT_ENOUGH_MEMORY – not enough memory to complete the operation
- ERR_INVALID_PARAMETER – can't allow the name of the enumeration value

Example:

```
enum interval  // enumeration of named constants
  {
   month=1,     // one-month interval
   two_months,  // two months
   quarter,     // three months - a quarter
   halfyear=6,  // half a year
   year=12,     // a year - 12 months
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the time interval equal to one month
   interval period=month;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- set the time interval equal to a quarter (three months)
   period=quarter;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- set the time interval equal to one year (12 months)
   period=year;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- check how the order type is shown
   ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;
   Print(EnumToString(type)+"="+IntegerToString(type));

//--- check how incorrect values are shown
   type=WRONG_VALUE;
   Print(EnumToString(type)+"="+IntegerToString(type));

// Result:
// month=1
// quarter=3
// year=12
// ORDER_TYPE_BUY=0
// ENUM_ORDER_TYPE::-1=-1
  }
```

See also

[Enumerations](), [Input variables]()

IntegerToString

This function converts value of integer type into a string of a specified length and returns the obtained string.

```
string  IntegerToString(
   long     number,              // number
   int      str_len=0,           // length of result string
   ushort   fill_symbol=' '      // filler
   );
```

Parameters

number

[in] Number for conversion.

str_len=0

[in] String length. If the resulting string length is larger than the specified one, the string is not cut off. If it is smaller, filler symbols will be added to the left.

fill_symbol=' '

[in] Filler symbol. By default it is a space.

Return Value

String.

Example:

```
#define DATA_TOTAL 1001

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- display rows with the index multiple of 100 in the loop by DATA_TOTAL
//--- the string displays the index value in the four-character format
//--- if the string length is less than 4 characters, then the value of the loop index
//--- in the string is preceded by leading zeros
   for(int i=0; i<DATA_TOTAL; i++)
     {
      if(i%100==0)
         Print("Converted index value: ",IntegerToString(i,4,'0'));
     }
   /*
   result:
   Converted index value: 0000
   Converted index value: 0100
   Converted index value: 0200
   Converted index value: 0300
   Converted index value: 0400
   Converted index value: 0500
   Converted index value: 0600
   Converted index value: 0700
   Converted index value: 0800
   Converted index value: 0900
   Converted index value: 1000
   */
  }
```

See also

StringToInteger

ShortToString

It converts the symbol code (unicode) into one-symbol string and returns resulting string.

```
string  ShortToString(
   ushort  symbol_code       // symbol
   );
```

Parameters

symbol_code

[in] Symbol code. Instead of a symbol code you can use literal string containing a symbol or a literal string with 2-byte hexadecimal code corresponding to the symbol from the Unicode table.

Return Value

String.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print 18 characters in a loop, starting with the character with the Unicode num
   for(int i=0; i<18; i++)
     {
      ushort code=0x23E9+(ushort)i;
      PrintFormat("Unicode number U+%hX: %s",code,ShortToString(code));
     }
   /*
   result:
   Unicode number U+23E9: ⏩
   Unicode number U+23EA: ⏪
   Unicode number U+23EB: ⏫
   Unicode number U+23EC: ⏬
   Unicode number U+23ED: ⏭
   Unicode number U+23EE: ⏮
   Unicode number U+23EF: ⏯
   Unicode number U+23F0: ⏰
   Unicode number U+23F1: ⏱
   Unicode number U+23F2: ⏲
   Unicode number U+23F3: ⏳
   Unicode number U+23F4: □
   Unicode number U+23F5: □
   Unicode number U+23F6: □
   Unicode number U+23F7: □
   Unicode number U+23F8: ⏸
```

```
    Unicode number U+23F9: ⏹
    Unicode number U+23FA: ⏺
    */
  }
```

See also

StringToCharArray, CharToString, StringGetCharacter

ShortArrayToString

It copies part of array into a returned string.

```
string  ShortArrayToString(
   ushort  array[],        // array
   int     start=0,        // starting position in the array
   int     count=-1        // number of symbols
   );
```

Parameters

array[]

[in] Array of ushort type (analog of wchar_t type).

start=0

[in] Position, from which copying starts, Default - 0.

count=-1

[in] Number of array elements to copy. Defines the length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.

Return Value

String.

Example:

```
#define ROW_SIZE 16

ushort ExtShortArray[]={0x2190,0x2191,0x2192,0x2193,0x2194,0x2195,0x2196,0x2197,0x219
                        0x21A0,0x21A1,0x21A2,0x21A3,0x21A4,0x21A5,0x21A6,0x21A7,0x21A
                        0x21B0,0x21B1,0x21B2,0x21B3,0x21B4,0x21B5,0x21B6,0x21B7,0x21B
                        0x21C0,0x21C1,0x21C2,0x21C3,0x21C4,0x21C5,0x21C6,0x21C7,0x21C
                        0x21D0,0x21D1,0x21D2,0x21D3,0x21D4,0x21D5,0x21D6,0x21D7,0x21D
                        0x21E0,0x21E1,0x21E2,0x21E3,0x21E4,0x21E5,0x21E6,0x21E7,0x21E
                        0x21F0,0x21F1,0x21F2,0x21F3,0x21F4,0x21F5,0x21F6,0x21F7,0x21F

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- calculate the number of strings in the symbol table of 7x16
   int total=(int)ExtShortArray.Size()/ROW_SIZE;
//--- in a loop by the table rows, display 7 sets of arrows, 16 pieces in each row in
   for(int i=0; i<total; i++)
     {
      int row=i*ROW_SIZE;
      PrintFormat("Arrow set %u: %s",i+1,ShortArrayToString(ExtShortArray,row,ROW_SIZ
     }
   /*
   result:
   Arrow set 1: ←↑→↓↔↕□□□□□□□ □ □□
   Arrow set 2: □□□□□□□□↨□□□□□□□
   Arrow set 3: □□□□□□□□□□□□□□□□
   Arrow set 4: □□□□□□□□□□□□□□□□
   Arrow set 5: □ □⇒ □⇔ □□□□□□□□□□□
   Arrow set 6: □□□□□□□□□□□□□□□□
   Arrow set 7: □□□□□□□□□□□□□□□□
   */
  }
```

See also

[StringToShortArray](), [CharArrayToString](), [Use of a Codepage]()

TimeToString

Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format.

```
string  TimeToString(
   datetime  value,                          // number
   int       mode=TIME_DATE|TIME_MINUTES      // output format
   );
```

Parameters

value

[in] Time in seconds from 00:00 1970/01/01.

mode=TIME_DATE|TIME_MINUTES

[in] Additional data input mode. Can be one or combined flag:
TIME_DATE gets result as "yyyy.mm.dd",
TIME_MINUTES gets result as "hh:mi",
TIME_SECONDS gets results as "hh:mi:ss".

Return Value

String.

Example:

```
datetime ExtBarTimeOpen;

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set the timer to one second
   EventSetTimer(1);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator deinitialization function                       |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   Comment("");
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- get the current bar opening time
   ExtBarTimeOpen=time[rates_total-1];
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- set the previous bar opening time
   static datetime bar_open_time=ExtBarTimeOpen;
//--- count the number of seconds passed since the bar opened
   static int seconds=int(TimeCurrent()-ExtBarTimeOpen);
//--- if the previous opening time is not equal to the current one, then this is a ne
//--- write the new opening time as the previous one and set the seconds to zero
   if(bar_open_time!=ExtBarTimeOpen)
     {
      bar_open_time=ExtBarTimeOpen;
      seconds=0;
     }
//--- increase and adjust the number of seconds passed since the bar opened
   seconds++;
   if(seconds>PeriodSeconds(PERIOD_CURRENT))
      seconds=0;
//--- bar opening time as yyyy.mm.dd hh:mi
   string bar_time_open=TimeToString(ExtBarTimeOpen);
//--- current time as yyyy.mm.dd hh:mi:ss
   string time_current=TimeToString(TimeCurrent(),TIME_DATE|TIME_MINUTES|TIME_SECONDS)
//--- number of seconds left until a new bar opens
   int    sec_left=PeriodSeconds(PERIOD_CURRENT)-seconds;
//--- amount of time remaining until the current bar closes as hh:mm:ss
   string time_left=TimeToString(sec_left,TIME_MINUTES|TIME_SECONDS);
//--- create the output string
   string txt=StringFormat("Opening time of the current bar: %s\n"+
                           "Time Current: %s\n"+
                           "Seconds have passed since the bar opened: %d\n"+
                           "Approximately seconds left before bar closes: %d\n"+
                           "Time remaining until bar closes: %s",bar_time_open,time_c
//--- display the bar open time and the current time,
//--- the number of seconds passed since the current bar opened and remaining until it
//--- the time remaining until the current bar closes in the comment
   Comment(txt);
   /*
   result on M1:
   Opening time of the current bar: 2024.02.22 18:06
   Time Current: 2024.02.22 18:06:24
   Seconds have passed since the bar opened: 25
   Approximately seconds left before bar closes: 35
   Time remaining until bar closes: 00:00:35

   result on M5:
   Opening time of the current bar: 2024.02.22 18:05
   Time Current: 2024.02.22 18:07:28
   Seconds have passed since the bar opened: 149
   Approximately seconds left before bar closes: 151
   Time remaining until bar closes: 00:02:31

   result on H1:
   Opening time of the current bar: 2024.02.22 18:00
   Time Current: 2024.02.22 18:08:13
   Seconds have passed since the bar opened: 494
   Approximately seconds left before bar closes: 3106
   Time remaining until bar closes: 00:51:46

   result on D1:
   Opening time of the current bar: 2024.02.22 00:00
   Time Current: 2024.02.22 18:11:01
   Seconds have passed since the bar opened: 65462
   Approximately seconds left before bar closes: 20938
   Time remaining until bar closes: 05:48:58
   */
  }
```

See also

StringToTime, TimeToStruct

NormalizeDouble

Rounding floating point number to a specified accuracy.

```
double  NormalizeDouble(
   double  value,      // normalized number
   int     digits      // number of digits after decimal point
   );
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format, number of digits after point (0-8).

Return Value

Value of double type with preset accuracy.

Note

Calculated values of StopLoss, TakeProfit, and values of open prices for pending orders must be normalized with the accuracy, the value of which can be obtained by Digits().

Please note that when output to Journal using the Print() function, a normalized number may contain a greater number of decimal places than you expect. For example, for:

```
   double a=76.671;             // A normalized number with three decimal places
   Print("Print(76.671)=",a);   // Output as is
   Print("DoubleToString(a,8)=",DoubleToString(a,8)); // Output with a preset accuracy
```

you will have the following in the terminal:

```
DoubleToString(a,8)=76.67100000

Print(76.671)=76.67100000000001
```

Example:

```
   double pi=M_PI;
   Print("pi = ",DoubleToString(pi,16));

   double pi_3=NormalizeDouble(M_PI,3);
   Print("NormalizeDouble(pi,3) = ",DoubleToString(pi_3,16))
   ;
   double pi_8=NormalizeDouble(M_PI,8);
   Print("NormalizeDouble(pi,8) = ",DoubleToString(pi_8,16));

   double pi_0=NormalizeDouble(M_PI,0);
   Print("NormalizeDouble(pi,0) = ",DoubleToString(pi_0,16));
```

```
/*
   Result:
   pi= 3.1415926535897931
   NormalizeDouble(pi,3)= 3.1419999999999999
   NormalizeDouble(pi,8)= 3.1415926499999998
   NormalizeDouble(pi,0)= 3.0000000000000000
*/
```

See also

DoubleToString, Real types (double, float), Typecasting

StringToCharArray

Symbol-wise copies a string converted from Unicode to ANSI, to a selected place of array of uchar type. It returns the number of copied elements.

```
int  StringToCharArray(
   string   text_string,          // source string
   uchar&   array[],              // array
   int      start=0,              // starting position in the array
   int      count=-1              // number of symbols
   uint     codepage=CP_ACP       // code page
   );
```

Parameters

text_string

[in] String to copy.

array[]

[out] Array of uchar type.

start=0

[in] Position from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0. Terminal 0 will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

codepage=CP_ACP

[in] The value of the code page. For the most-used code pages provide appropriate constants.

Return Value

Number of copied elements.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- string to convert
   string text = "This is a test of the StringToCharArray() function";

//--- convert the input string into a uchar array in accordance with the set code page
   uchar char_array[];
   int   copied=StringToCharArray(text, char_array);
   PrintFormat("String length: %u\nNumber of characters copied (with terminal 0): %d\
               StringLen(text), copied, text);
//--- print the resulting array to the journal
   ArrayPrint(char_array, 0, " | ");
   /*
   result:
   String length: 50
   Number of characters copied (with terminal 0): 51
   Array of characters for the string 'This is a test of the StringToCharArray() funct
   [ 0]  84 | 104 | 105 | 115 |  32 | 105 | 115 |  32 |  97 |  32 | 116 | 101 | 115 |
   [17]  32 | 116 | 104 | 101 |  32 |  83 | 116 | 114 | 105 | 110 | 103 |  84 | 111 |
   [34]  65 | 114 | 114 |  97 | 121 |  40 |  41 |  32 | 102 | 117 | 110 |  99 | 116 |
   */
  }
```

See also

CharArrayToString, StringToShortArray, Use of a Codepage

StringToColor

Converting "R,G,B" string or string with color name into color type value.

```
color  StringToColor(
   string  color_string      // string representation of color
   );
```

Parameters

color_string

[in] String representation of a color of "R,G,B" type or name of one of predefined Web-colors.

Return Value

Color value.

Example:

```
   color str_color=StringToColor("0,127,0");
   Print(str_color);
   Print((string)str_color);
//--- change color a little
   str_color=StringToColor("0,128,0");
   Print(str_color);
   Print((string)str_color);
```

See also

ColorToString, ColorToARGB

StringToDouble

The function converts string containing a symbol representation of number into number of double type.

```
double  StringToDouble(
   string  value       // string
   );
```

Parameters

value

[in] String containing a symbol representation of a number.

Return Value

Value of double type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- string to convert
   string str = "12345.54321";
//--- convert the input string as a real number into a double type variable
   double converted=StringToDouble(str);
//--- display the resulting number accurate to 8 decimal places in the journal
   PrintFormat("The string '%s' is converted to the real number %.8f", str, converted)
   /*
   result:
   The string '12345.54321' is converted to the real number 12345.54321000
   */
  }
```

See also

NormalizeDouble, Real types (double, float), Typecasting

StringToInteger

The function converts string containing a symbol representation of number into number of long (integer) type.

```
long  StringToInteger(
   string  value       // string
   );
```

Parameters

value

[in] String containing a number.

Return Value

Value of long type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- string to convert
   string str = "12345.54321";
//--- convert the input string as a number into a long type variable
   long converted=StringToInteger(str);
//--- display the obtained number in the journal with the fractional part cut off
   PrintFormat("The string '%s' is converted to the integer number %I64d", str, conver
   /*
   result:
   The string '12345.54321' is converted to the integer number 12345
   */
  }
```

See also

[IntegerToString](), [Real types (double, float)](), [Typecasting]()

StringToShortArray

The function symbol-wise copies a string into a specified place of an array of ushort type. It returns the number of copied elements.

```
int  StringToShortArray(
   string  text_string,      // source string
   ushort& array[],          // array
   int     start=0,          // starting position in the array
   int     count=-1          // number of symbols
   );
```

Parameters

text_string

[in] String to copy

array[]

[out] Array of ushort type (analog of wchar_t type).

start=0

[in] Position, from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.Terminal 0 will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

Return Value

Number of copied elements.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- string with chars to convert
   string text = "Chars: ♡♫☎✈✢ ☏ ☼✉☆ ☁♕ ♠®✧ ❄ ♣   ";
   ushort short_array[];
//--- find the position of the ":" character in the input string and extract the subst
   int    pos=StringFind(text, ":");
   string str=(pos<0 ? text : StringSubstr(text, pos+1));
//--- remove spaces, carriage movement and tabulation characters from the left and ri
   StringTrimLeft(str);
   StringTrimRight(str);
//--- copy the resulting string into the ushort array and print the resulting array to
   int copied=StringToShortArray(str, short_array);
   PrintFormat("String of characters length: %u\n"
               "Number of characters copied (with terminal 0): %d\n"
               "Array of chars for the string '%s':",
               StringLen(str), copied, str);
   ArrayPrint(short_array, 0, " | ");
   /*
   result:
   String of characters length: 16
   Number of characters copied (with terminal 0): 17
   Array of chars for the string '❤♫☎✈✣☏☀✉☆☁♕♠®✧❄♣':
   10084 |  9835 |  9742 |  9992 | 10019 |  9743 |  9728 |  9993 |  9734 |  9729 |  98
   */
  }
```

See also

ShortArrayToString, StringToCharArray, Use of a Codepage

StringToTime

Transforms the string containing time and/or date in the "yyyy.mm.dd [hh:mi]" format into the datetime type number.

```
datetime  StringToTime(
   const string  time_string      // date string
   );
```

Parameters

time_string

[in] String in one of the specified formats:

- "yyyy.mm.dd [hh:mi]"
- "yyyy.mm.dd [hh:mi:ss]"
- "yyyymmdd [hh:mi:ss]"
- "yyyymmdd [hhmiss]"
- "yyyy/mm/dd [hh:mi:ss]"
- "yyyy-mm-dd [hh:mi:ss]"

Return Value

datetime type value containing the number of seconds elapsed since 01.01.1970.

Note

Any sequence of space and tabulation characters between date and time is considered to be a single space to avoid additional processing of the *time_string* before calling StringToTime().

Example:

```
//--- input parameters
input group    "The date can be entered in any of the formats:"
input group    "yyyy.mm.dd [hh:mi], yyyy.mm.dd [hh:mi:ss]"
input group    "yyyymmdd [hh:mi:ss], yyyymmdd [hhmiss]"
input group    "yyyy/mm/dd [hh:mi:ss], yyyy-mm-dd [hh:mi:ss]"
input string   InpDateStr;     // Please enter the date here as a string

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- convert the time entered in the inputs as a string into a datetime value
   datetime time=StringToTime(InpDateStr);
//--- display the entered string and the obtained time in the journal
   PrintFormat("Date entered as a string in the form '%s' is converted to datetime in
                InpDateStr, TimeToString(time, TIME_DATE|TIME_MINUTES|TIME_SECONDS));
//--- create a vertical line on the received date-time and shift the chart to this loc
   if(CreateVLine(time))
      ChartNavigateToTime(time);
   /*
   result:
   Date entered as a string in the form '' is converted to datetime in the form '1970.
   Date entered as a string in the form '2024' is converted to datetime in the form '2
   Date entered as a string in the form '202400' is converted to datetime in the form
   Date entered as a string in the form '20240000' is converted to datetime in the for
   Date entered as a string in the form '2024022410' is converted to datetime in the f
   Date entered as a string in the form '20240224 10' is converted to datetime in the
   Date entered as a string in the form '20240224 01' is converted to datetime in the
   Date entered as a string in the form '20240224 0030' is converted to datetime in th
   Date entered as a string in the form '20240224 0100' is converted to datetime in th
   */
  }
//+------------------------------------------------------------------+
//| Create a vertical line object                                    |
//+------------------------------------------------------------------+
bool CreateVLine(const datetime line_time)
  {
   ResetLastError();

   string name=MQLInfoString(MQL_PROGRAM_NAME)+"_VLINE";
   if(!ObjectCreate(0, name, OBJ_VLINE, 0, line_time, 0))
     {
      Print("ObjectCreate() failed. Error code: ", GetLastError());
      return(false);
     }
   ObjectSetInteger(0, name, OBJPROP_STYLE, STYLE_DOT);
   ObjectSetInteger(0, name, OBJPROP_SELECTABLE, true);

   return(true);
  }
//+------------------------------------------------------------------+
//| Shift the chart to the specified bar opening time                |
//+------------------------------------------------------------------+
bool ChartNavigateToTime(const datetime time)
  {
   ChartSetInteger(0, CHART_AUTOSCROLL, false);
   ResetLastError();

   int bar=iBarShift(_Symbol, PERIOD_CURRENT, time);
   if(bar<0)
     {
      PrintFormat("%s: iBarShift() failed. Error code: %d", __FUNCTION__, GetLastErro
      return(false);
     }
   long first=0;
   if(!ChartGetInteger(0, CHART_FIRST_VISIBLE_BAR, 0, first))
     {
      PrintFormat("%s: ChartGetInteger() failed. Error code: %d", __FUNCTION__, GetLas
      return(false);
     }

   return(ChartNavigate(0, CHART_CURRENT_POS, (int)first-bar));
  }
```

See also

TimeToString, TimeToStruct

StringFormat

The function formats obtained parameters and returns a string.

```
string  StringFormat(
   string   format,      // string with format description
   ...      ...          // parameters
   );
```

Parameters

format

[in] String containing method of formatting. Formatting rules are the same as for the PrintFormat function.

...

[in] Parameters, separated by a comma.

Return Value

String.

Example:

```
void OnStart()
  {
//--- string variables
   string output_string;
   string temp_string;
   string format_string;
//--- prepare the specification header
   temp_string=StringFormat("Contract specification for %s:\n",_Symbol);
   StringAdd(output_string,temp_string);
//--- int value output
   int digits=(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   temp_string=StringFormat("   SYMBOL_DIGITS = %d (number of digits after the decimal
                            digits);
   StringAdd(output_string,temp_string);
//--- double value output with variable number of digits after the decimal point
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   format_string=StringFormat("   SYMBOL_POINT = %%.%df (point value)\n",
                              digits);
   temp_string=StringFormat(format_string,point_value);
   StringAdd(output_string,temp_string);
//--- int value output
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   temp_string=StringFormat("   SYMBOL_SPREAD = %d (current spread in points)\n",
                            spread);
   StringAdd(output_string,temp_string);
//--- int value output
   int min_stop=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
   temp_string=StringFormat("   SYMBOL_TRADE_STOPS_LEVEL = %d (minimal indention in po
                            min_stop);
   StringAdd(output_string,temp_string);
//--- double value output without the fractional part
   double contract_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_CONTRACT_SIZE);
   temp_string=StringFormat("   SYMBOL_TRADE_CONTRACT_SIZE = %.f (contract size)\n",
                            contract_size);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double tick_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_TICK_SIZE);
   temp_string=StringFormat("   SYMBOL_TRADE_TICK_SIZE = %f (minimal price change)\n",
                            tick_size);
   StringAdd(output_string,temp_string);
//--- determining the swap calculation mode
   int swap_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_SWAP_MODE);
   string str_swap_mode;
   switch(swap_mode)
     {
      case SYMBOL_SWAP_MODE_DISABLED: str_swap_mode="SYMBOL_SWAP_MODE_DISABLED (no swa
      case SYMBOL_SWAP_MODE_POINTS: str_swap_mode="SYMBOL_SWAP_MODE_POINTS (in points)
      case SYMBOL_SWAP_MODE_CURRENCY_SYMBOL: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
      case SYMBOL_SWAP_MODE_CURRENCY_MARGIN: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_
      case SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY
      case SYMBOL_SWAP_MODE_INTEREST_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_INTERES
      case SYMBOL_SWAP_MODE_INTEREST_OPEN: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST_O
      case SYMBOL_SWAP_MODE_REOPEN_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_CU
      case SYMBOL_SWAP_MODE_REOPEN_BID: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_BID (b
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_SWAP_MODE = %s\n",
                            str_swap_mode);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double swap_long=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_LONG);
   temp_string=StringFormat("   SYMBOL_SWAP_LONG = %f (long swap value)\n",
                            swap_long);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double swap_short=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_SHORT);
   temp_string=StringFormat("   SYMBOL_SWAP_SHORT = %f (short swap value)\n",
                            swap_short);
   StringAdd(output_string,temp_string);
//--- determining the trading mode
   int trade_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_MODE);
   string str_trade_mode;
   switch(trade_mode)
     {
      case SYMBOL_TRADE_MODE_DISABLED: str_trade_mode="SYMBOL_TRADE_MODE_DISABLED (tra
      case SYMBOL_TRADE_MODE_LONGONLY: str_trade_mode="SYMBOL_TRADE_MODE_LONGONLY (onl
      case SYMBOL_TRADE_MODE_SHORTONLY: str_trade_mode="SYMBOL_TRADE_MODE_SHORTONLY (c
      case SYMBOL_TRADE_MODE_CLOSEONLY: str_trade_mode="SYMBOL_TRADE_MODE_CLOSEONLY (c
      case SYMBOL_TRADE_MODE_FULL: str_trade_mode="SYMBOL_TRADE_MODE_FULL (no trade re
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_TRADE_MODE = %s\n",
                            str_trade_mode);
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   temp_string=StringFormat("   SYMBOL_VOLUME_MIN = %g (minimal volume for a deal)\n",
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_step=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_STEP);
   temp_string=StringFormat("   SYMBOL_VOLUME_STEP = %g (minimal volume change step)\n
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_max=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MAX);
   temp_string=StringFormat("   SYMBOL_VOLUME_MAX = %g (maximal volume for a deal)\n",
   StringAdd(output_string,temp_string);
//--- determining the contract price calculation mode
   int calc_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_CALC_MODE);
   string str_calc_mode;
   switch(calc_mode)
     {
      case SYMBOL_CALC_MODE_FOREX:str_calc_mode="SYMBOL_CALC_MODE_FOREX (Forex)";break
      case SYMBOL_CALC_MODE_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_FUTURES (futures)"
      case SYMBOL_CALC_MODE_CFD:str_calc_mode="SYMBOL_CALC_MODE_CFD (CFD)";break;
      case SYMBOL_CALC_MODE_CFDINDEX:str_calc_mode="SYMBOL_CALC_MODE_CFDINDEX (CFD for
      case SYMBOL_CALC_MODE_CFDLEVERAGE:str_calc_mode="SYMBOL_CALC_MODE_CFDLEVERAGE (C
      case SYMBOL_CALC_MODE_EXCH_STOCKS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_STOCKS (t
      case SYMBOL_CALC_MODE_EXCH_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FUTURES 
      case SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FU
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_TRADE_CALC_MODE = %s\n",
                            str_calc_mode);
   StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
   double margin_initial=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_INITIAL);
   temp_string=StringFormat("   SYMBOL_MARGIN_INITIAL = %.2f (initial margin)\n",
                            margin_initial);
   StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
   double margin_maintenance=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_MAINTENANCE);
   temp_string=StringFormat("   SYMBOL_MARGIN_MAINTENANCE = %.2f (maintenance margin)\
                            margin_maintenance);
   StringAdd(output_string,temp_string);
//--- int value output
   int freeze_level=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_FREEZE_LEVEL);
   temp_string=StringFormat("   SYMBOL_TRADE_FREEZE_LEVEL = %d (order freeze level in
                            freeze_level);
   StringAdd(output_string,temp_string);
   Print(output_string);
   Comment(output_string);
/* execution result
   Contract specification for EURUSD:
     SYMBOL_DIGITS = 5 (number of digits after the decimal point)
     SYMBOL_POINT = 0.00001 (point value)
     SYMBOL_SPREAD = 10 (current spread in points)
     SYMBOL_TRADE_STOPS_LEVEL = 18 (minimal indention in points for Stop orders)
     SYMBOL_TRADE_CONTRACT_SIZE = 100000 (contract size)
     SYMBOL_TRADE_TICK_SIZE = 0.000010 (minimal price change)
     SYMBOL_SWAP_MODE = SYMBOL_SWAP_MODE_POINTS (in points)
     SYMBOL_SWAP_LONG = -0.700000 (buy order swap value)
     SYMBOL_SWAP_SHORT = -1.000000 (sell order swap value)
     SYMBOL_TRADE_MODE = SYMBOL_TRADE_MODE_FULL (no trade restrictions)
     SYMBOL_VOLUME_MIN = 0.01 (minimal volume for a deal)
     SYMBOL_VOLUME_STEP = 0.01 (minimal volume change step)
     SYMBOL_VOLUME_MAX = 500 (maximal volume for a deal)
     SYMBOL_TRADE_CALC_MODE = SYMBOL_CALC_MODE_FOREX (Forex)
     SYMBOL_MARGIN_INITIAL = 0.00 (initial margin)
     SYMBOL_MARGIN_MAINTENANCE = 0.00 (maintenance margin)
     SYMBOL_TRADE_FREEZE_LEVEL = 0 (order freeze level in points)
*/
  }
```

See also

PrintFormat, DoubleToString,ColorToString, TimeToString

Mathematical Functions

A set of mathematical and trigonometric functions.

Math functions were originally designed to perform relevant operations on scalar values. From this build on, most of the functions can be applied to matrices and vectors. These include MathAbs, MathArccos, MathArcsin, MathArctan, MathCeil, MathCos, MathExp, MathFloor, MathLog, MathLog10, MathMod, MathPow, MathRound, MathSin, MathSqrt, MathTan, MathExpm1, MathLog1p, MathArccosh, MathArcsinh, MathArctanh, MathCosh, MathSinh, and MathTanh. Such operations imply element-wise handling of matrices or vectors. Example:

```
//---
   matrix a= {{1, 4}, {9, 16}};
   Print("matrix a=\n",a);
   a=MathSqrt(a);
   Print("MatrSqrt(a)=\n",a);
   /*
    matrix a=
    [[1,4]
     [9,16]]
    MatrSqrt(a)=
    [[1,2]
     [3,4]]
   */
```

For MathMod and MathPow, the second element can be either a scalar or a matrix/vector of the appropriate size.

Function	Action
MathAbs	Returns absolute value (modulus) of the specified numeric value
MathArccos	Returns the arc cosine of x in radians
MathArcsin	Returns the arc sine of x in radians
MathArctan	Returns the arc tangent of x in radians
MathArctan2	Return the angle (in radians) whose tangent is the quotient of two specified numbers
MathClassify	Returns the type of a real number
MathCeil	Returns integer numeric value closest from above
MathCos	Returns the cosine of a number
MathExp	Returns exponent of a number
MathFloor	Returns integer numeric value closest from below
MathLog	Returns natural logarithm
MathLog10	Returns the logarithm of a number by base 10

Function	Action
MathMax	Returns the maximal value of the two numeric values
MathMin	Returns the minimal value of the two numeric values
MathMod	Returns the real remainder after the division of two numbers
MathPow	Raises the base to the specified power
MathRand	Returns a pseudorandom value within the range of 0 to 32767
MathRound	Rounds of a value to the nearest integer
MathSin	Returns the sine of a number
MathSqrt	Returns a square root
MathSrand	Sets the starting point for generating a series of pseudorandom integers
MathTan	Returns the tangent of a number
MathIsValidNumber	Checks the correctness of a real number
MathExpm1	Returns the value of the expression MathExp(x)-1
MathLog1p	Returns the value of the expression MathLog(1+x)
MathArccosh	Returns the hyperbolic arccosine
MathArcsinh	Returns the hyperbolic arcsine
MathArctanh	Returns the hyperbolic arctangent
MathCosh	Returns the hyperbolic cosine
MathSinh	Returns the hyperbolic sine
MathTanh	Returns the hyperbolic tangent
MathSwap	Change the order of bytes in the ushort/uint/ushort types value

MathAbs

The function returns the absolute value (modulus) of the specified numeric value.

```
double  MathAbs(
   double  value       // numeric value
   );
```

Parameters

value

  [in] Numeric value.

Return Value

Value of double type more than or equal to zero.

Note

Instead the MathAbs() function you can use fabs().

Example:

```
//--- input parameters
input int       InpValue = -10;    // Enter any value here

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the absolute value of the number entered in the inputs
   uint abs_value=MathAbs(InpValue);
//--- print the values in the journal
   PrintFormat("The entered value %d received the value %u after using the MathAbs() f
  }
```

MathArccos

The function returns the arccosine of x within the range 0 to π in radians.

```
double  MathArccos(
   double  val      // -1<val<1
   );
```

Parameters

val

[in] The val value between -1 and 1, the arc cosine of which is to be calculated.

Return Value

Arc cosine of a number in radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

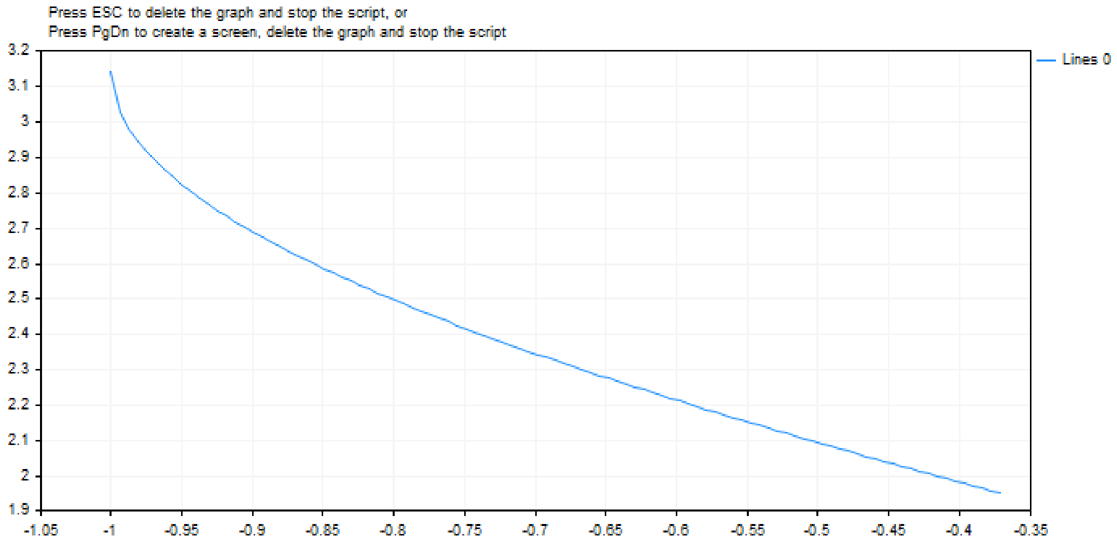
Instead of the MathArccos() function you can use acos().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/1000);
   delta[0]=-1;
//--- get 101 values from -1 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the arc cosine value for each value of the X vector
   vector Y=MathArccos(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

```

```
//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and saves the image to a file                  |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



Press ESC to delete the graph and stop the script, or
Press PgDn to create a screen, delete the graph and stop the script

3.2
3.1
3
2.9
2.8
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2
1.9
-1.05
-1
-0.95
-0.9
-0.85
-0.8
-0.75
-0.7
-0.65
-0.6
-0.55
-0.5
-0.45
-0.4
-0.35
Lines 0

See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArcsin

The function returns the arc sine of x within the range of -π/2 to π/2 radians.

```
double  MathArcsin(
   double  val       // -1<value<1
   );
```

Parameters

val

[in] The val value between -1 and 1, the arc sine of which is to be calculated.

Return Value

Arc sine of the val number in radians within the range of -π/2 to π/2 radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

Instead of the MathArcsin() function you can use asin().

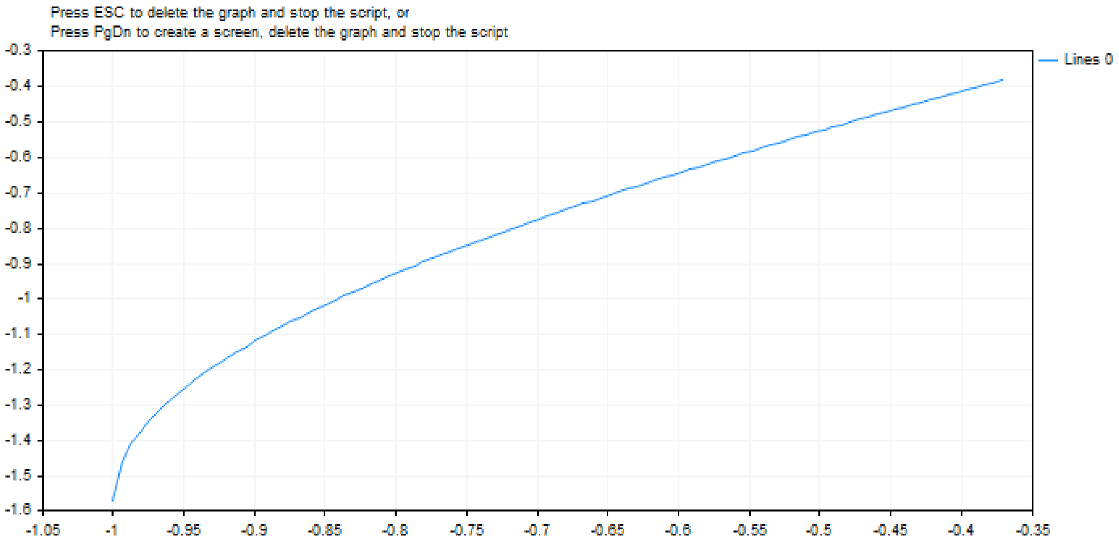
Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/1000);
   delta[0]=-1;
//--- get 101 values from -1 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the sine value for each value of the X vector
   vector Y=MathArcsin(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);
```

```
//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
```

```
 int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRIT
 if(selected<1)
   {
    if(selected<0)
       PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
    return false;
   }

 bool res=false;
 if(ChartSetInteger(0,CHART_SHOW,false))
    res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
 ChartSetInteger(0,CHART_SHOW,true);
 return(res);
}
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArctan

The function returns the arc tangent of x. If x is equal to 0, the function returns 0.

```
double  MathArctan(
   double  value      // tangent
   );
```

Parameters

value

[in] A number representing a tangent.

Return Value

MathArctan returns a value within the range of -π/2 to π/2 radians.

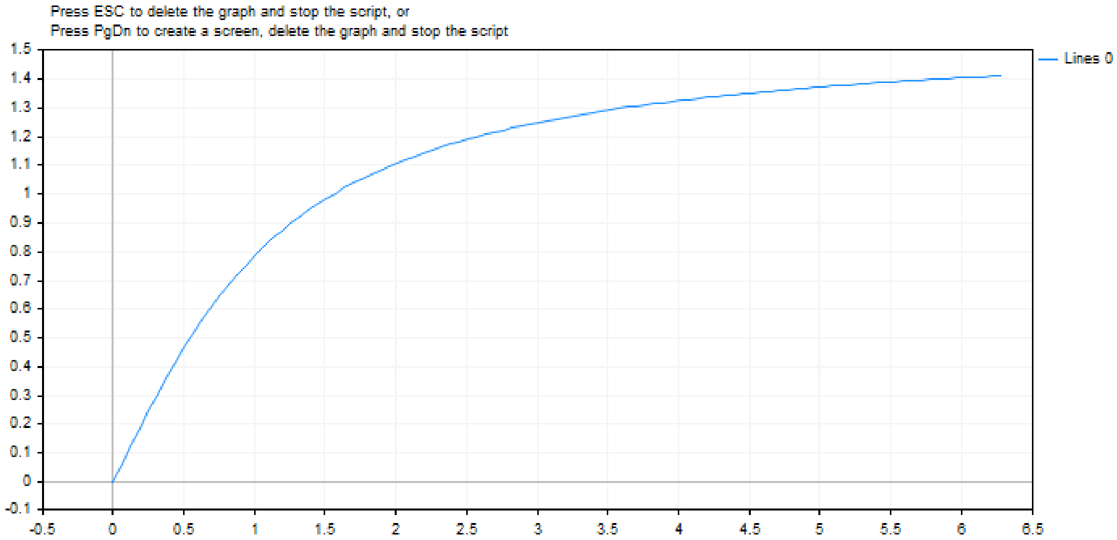
Note

Instead of the MathArctan() function you can use atan().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the arc tangent value for each value of the X vector
   vector Y=MathArctan(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
 if(selected<1)
   {
    if(selected<0)
       PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
    return false;
   }

 bool res=false;
 if(ChartSetInteger(0,CHART_SHOW,false))
    res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
 ChartSetInteger(0,CHART_SHOW,true);
 return(res);
}
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArctan2

Return the angle (in radians) whose tangent is the quotient of two specified numbers.

```
double MathArctan2(
 double y      // The y coordinate of a point
 double x      // The x coordinate of a point
  );
```

Parameters

y

[in] Y coordinate value.

x

[in] X coordinate value.

Return Value

MathArctan2 returns an angle, θ, within the range from -π to π radians, so that MathTan(θ)=y/x.

Please note as follows:

- For (x, y) in quadrant 1, 0< θ < π/2
- For (x, y) in quadrant 2, π/2< θ ≤ π
- For (x, y) in quadrant 3, -π < θ < -π/2
- For (x, y) in quadrant 4, -π/2 < θ < 0

For points on the boundaries of the quadrants, the return value is the following:

- If y is 0 and x is not negative, θ = 0.
- If y is 0 and x is negative, θ = π.
- If y is positive and x is 0, θ = π/2.
- If y is negative and x is 0, θ = -π/2.
- If y is 0 and x is 0, θ = 0.

Note

Instead of the MathArctan2() function, you can use the atan2() function.

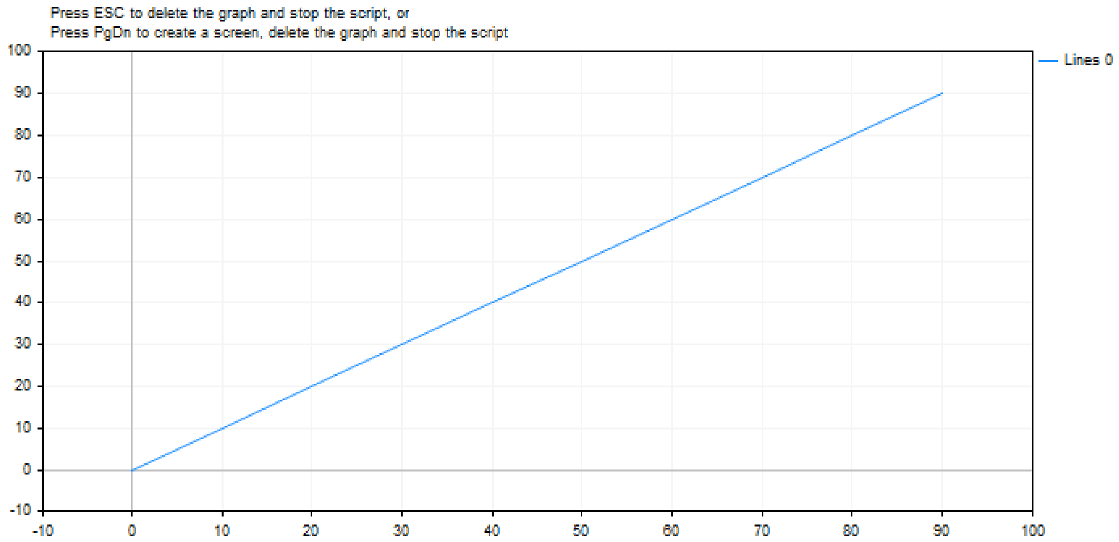
Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   vector delta=vector::Full(10,10);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the arc tangent value for each value of the X vector
   vector Y=delta.CumSum();

   Print("vector delta = \n",delta);
   Print("vector X = \n",X);
   Print("vector Y = \n",Y);

//--- transfer the calculated values from vectors to arrays
   double x_array[];;
   double y_array[];;
   X.Swap(x_array);
   Y.Swap(y_array);

   double array[10];
   for(int i=0; i<10; i++)
     {
      array[i]=MathArctan2(y_array[i],x_array[i]);
     }

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
```

```
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

Real types (double, float), Statistics, Scientific Charts, Client Terminal Properties

MathClassify

Determines the type of a real number and returns a result as a value from the ENUM_FP_CLASS enumeration

```
ENUM_FP_CLASS  MathClassify(
   double  value       // real number
   );
```

Parameters

value

[in] The real number to be checked

Return Value

A value from the ENUM_FP_CLASS enumeration

ENUM_FP_CLASS

ID	Description
FP_SUBNORMAL	A subnormal number which is closer to zero than the smallest representable normal number DBL_MIN (2.2250738585072014e-308)
FP_NORMAL	A normal number in the range between 2.2250738585072014e-308 and 1.7976931348623158e+308
FP_ZERO	A positive or a negative zero
FP_INFINITE	A number which cannot be represented by the appropriate type, positive or negative infinity
FP_NAN	Not a number.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- test NaN
  double nan=double("nan");
  PrintFormat("Test NaN: %G is %s, MathIsValidNumber(NaN)=%s",
              nan,
              EnumToString(MathClassify(nan)),
              (string)MathIsValidNumber(nan));
//--- test infinity
  double inf=double("inf");
  PrintFormat("Test Inf: %G is %s, MathIsValidNumber(inf)=%s",
              inf,
              EnumToString(MathClassify(inf)),
              (string)MathIsValidNumber(inf));
//--- test normal value
  double normal=1.2345e6;
  PrintFormat("Test Normal: %G is %s, MathIsValidNumber(normal)=%s",
              normal,
              EnumToString(MathClassify(normal)),
              (string)MathIsValidNumber(normal));
//--- test subnormal value
  double sub_normal=DBL_MIN/2.0;
  PrintFormat("Test Subnormal: %G is %s, MathIsValidNumber(sub_normal)=%s",
              sub_normal,
              EnumToString(MathClassify(sub_normal)),
              (string)MathIsValidNumber(sub_normal));
//--- test zero value
  double zero=0.0/(-1);
  PrintFormat("Test Zero: %G is %s, MathIsValidNumber(zero)=%s",
              zero,
              EnumToString(MathClassify(zero)),
              (string)MathIsValidNumber(zero));
 }
 /*
 Result:
   Test NaN: NAN is FP_NAN, MathIsValidNumber(NaN)=false
   Test Inf: INF is FP_INFINITE, MathIsValidNumber(inf)=false
   Test Normal: 1.2345E+06 is FP_NORMAL, MathIsValidNumber(normal)=true
   Test Subnormal: 1.11254E-308 is FP_SUBNORMAL, MathIsValidNumber(sub_normal)=true
   Test Zero: -0 is FP_ZERO, MathIsValidNumber(zero)=true
*/
//+------------------------------------------------------------------+
```

See also

Real types (double, float), MathIsValidNumber

MathCeil

The function returns integer numeric value closest from above.

```
double  MathCeil(
   double  val      // number
   );
```

Parameters

val

[in] Numeric value.

Return Value

Numeric value representing the smallest integer that exceeds or equals to val.

Note

Instead of the MathCeil() function you can use ceil().

Example:

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare variables for conversion
   double value=0;                          // real number for MathCeil conversion
   int    ceil_value=0;                     // get the result here
//--- in a loop by the number of decimal increments of a real number
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- increase the number value,
      //--- get the nearest integer value from above
      //--- and display control values in the journal
      value+=0.1;
      ceil_value=(int)MathCeil(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, ceil value: %d",value,ceil_value);
      /*
      result:
      value: 0.1, ceil value: 1
      value: 0.2, ceil value: 1
      value: 0.3, ceil value: 1
      value: 0.4, ceil value: 1
      value: 0.5, ceil value: 1
      value: 0.6, ceil value: 1
      value: 0.7, ceil value: 1
      value: 0.8, ceil value: 1
      value: 0.9, ceil value: 1
      value: 1.0, ceil value: 1
      value: 1.1, ceil value: 2
      value: 1.2, ceil value: 2
      value: 1.3, ceil value: 2
      value: 1.4, ceil value: 2
      value: 1.5, ceil value: 2
      value: 1.6, ceil value: 2
      value: 1.7, ceil value: 2
      value: 1.8, ceil value: 2
      value: 1.9, ceil value: 2
      value: 2.0, ceil value: 2
      value: 2.1, ceil value: 3
      value: 2.2, ceil value: 3
      value: 2.3, ceil value: 3
      value: 2.4, ceil value: 3
      value: 2.5, ceil value: 3
      value: 2.6, ceil value: 3
      value: 2.7, ceil value: 3
      value: 2.8, ceil value: 3
      value: 2.9, ceil value: 3
      value: 3.0, ceil value: 3
      value: 3.1, ceil value: 4
      */
     }
  }
```

MathCos

The function returns the cosine of an angle.

```
double  MathCos(
   double  value      // number
   );
```

Parameters

value

[in] Angle in radians.

Return Value

Value of double type within the range of -1 to 1.

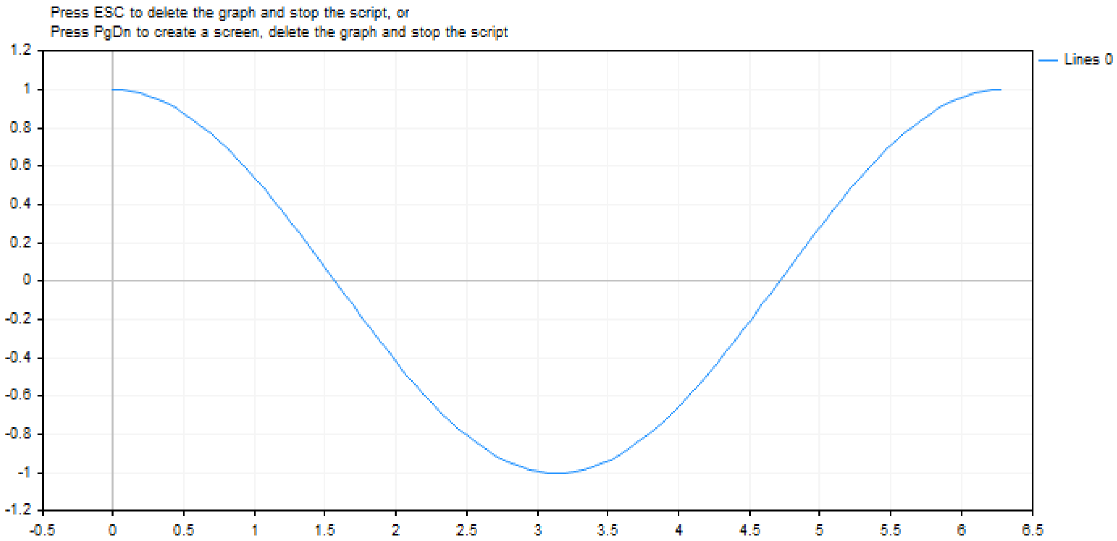
Note

Instead of MathCos() you can use cos().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the cosine value for each value of the X vector
   vector Y=MathCos(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathExp

The function returns the value of e raised to the power of d.

```
double  MathExp(
   double  value      // power for the number e
   );
```

Parameters

value

[in] A number specifying the power.

Return Value

A number of double type. In case of overflow the function returns INF (infinity), in case of underflow MathExp returns 0.

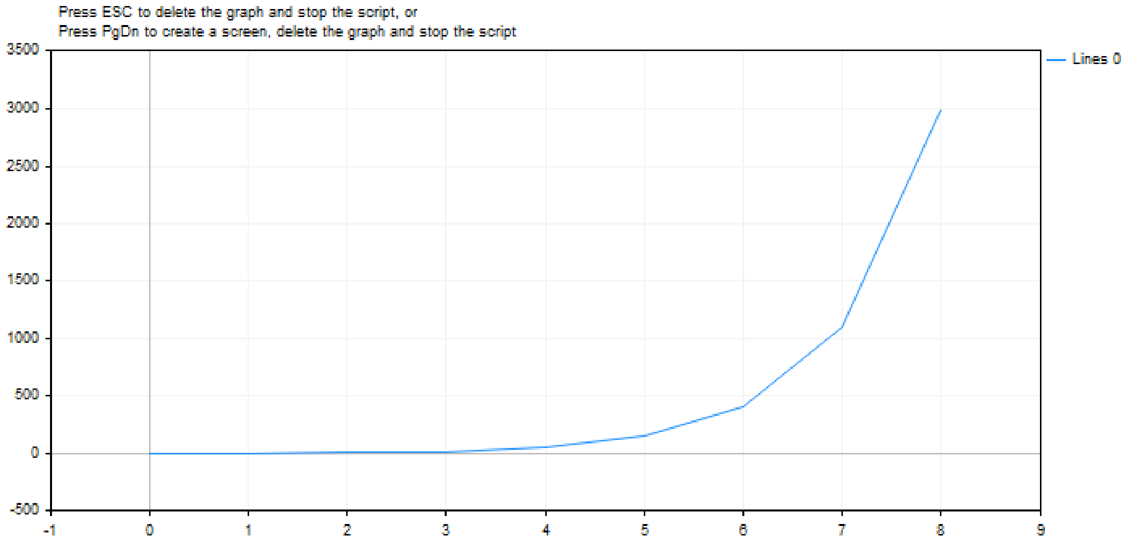
Note

Instead of MathExp() you can use exp().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 9 values from 0 to 8 with step 1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- calculate the "e" (Euler number) to the power of each X vector value
   X=MathExp(X);
   Print("MathExp(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathFloor

The function returns integer numeric value closest from below.

```
double  MathFloor(
   double  val      // number
   );
```

Parameters

val

[in] Numeric value.

Return Value

A numeric value representing the largest integer that is less than or equal to val.

Note

Instead of MathFloor() you can use floor().

Example:

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare variables for conversion
   double value=0;                       // real number for MathFloor conversion
   int    floor_value=0;                 // get the result here
//--- in a loop by the number of decimal increments of a real number
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- increase the number value,
      //--- get the nearest integer value from below
      //--- and display control values in the journal
      value+=0.1;
      floor_value=(int)MathFloor(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, floor value: %d",value,floor_value);
      /*
      result:
      value: 0.1, floor value: 0
      value: 0.2, floor value: 0
      value: 0.3, floor value: 0
      value: 0.4, floor value: 0
      value: 0.5, floor value: 0
      value: 0.6, floor value: 0
```

```
    value: 0.7, floor value: 0
    value: 0.8, floor value: 0
    value: 0.9, floor value: 0
    value: 1.0, floor value: 1
    value: 1.1, floor value: 1
    value: 1.2, floor value: 1
    value: 1.3, floor value: 1
    value: 1.4, floor value: 1
    value: 1.5, floor value: 1
    value: 1.6, floor value: 1
    value: 1.7, floor value: 1
    value: 1.8, floor value: 1
    value: 1.9, floor value: 1
    value: 2.0, floor value: 2
    value: 2.1, floor value: 2
    value: 2.2, floor value: 2
    value: 2.3, floor value: 2
    value: 2.4, floor value: 2
    value: 2.5, floor value: 2
    value: 2.6, floor value: 2
    value: 2.7, floor value: 2
    value: 2.8, floor value: 2
    value: 2.9, floor value: 2
    value: 3.0, floor value: 3
    value: 3.1, floor value: 3
    */
   }
```

MathLog

The function returns a natural logarithm.

```
double  MathLog(
   double  val       // value to take the logarithm
   );
```

Parameters

val

[in] Value logarithm of which is to be found.

Return Value

The natural logarithm of val in case of success. If val is negative, the function returns NaN (undetermined value). If val is equal to 0, the function returns INF (infinity).

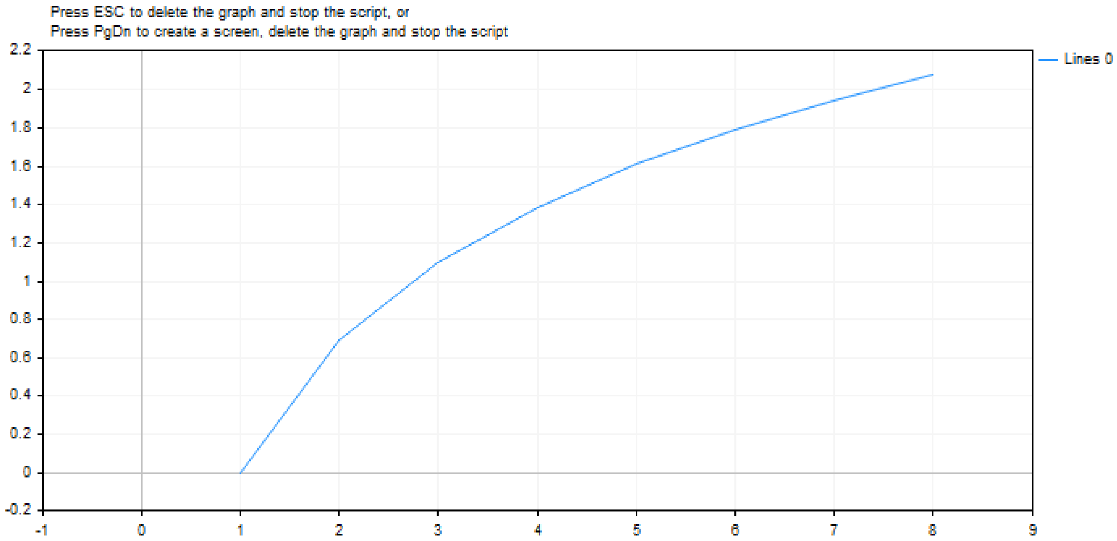
Note

Instead of MathLog() you can use log().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 9 values from 0 to 8 with step 1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- calculate the logarithm for each value of the X vector
   X=MathLog(X);
   Print("MathLog(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
   /*
   result:
   vector X = 
   [0,1,2,3,4,5,6,7,8]
   MathLog(X) = 
   [-inf,0,0.6931471805599453,1.09861228866811,1.386294361119891,1.6094379124341,1.79
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

Real types (double, float), Statistics, Scientific Charts, Client Terminal Properties

MathLog

Returns the logarithm of a number by base 10.

```
double  MathLog10(
   double  val      // number to take logarithm
   );
```

Parameters

val

[in] Numeric value the common logarithm of which is to be calculated.

Return Value

The common logarithm in case of success. If val is negative, the function returns NaN (undetermined value). If val is equal to 0, the function returns INF (infinity).

Note

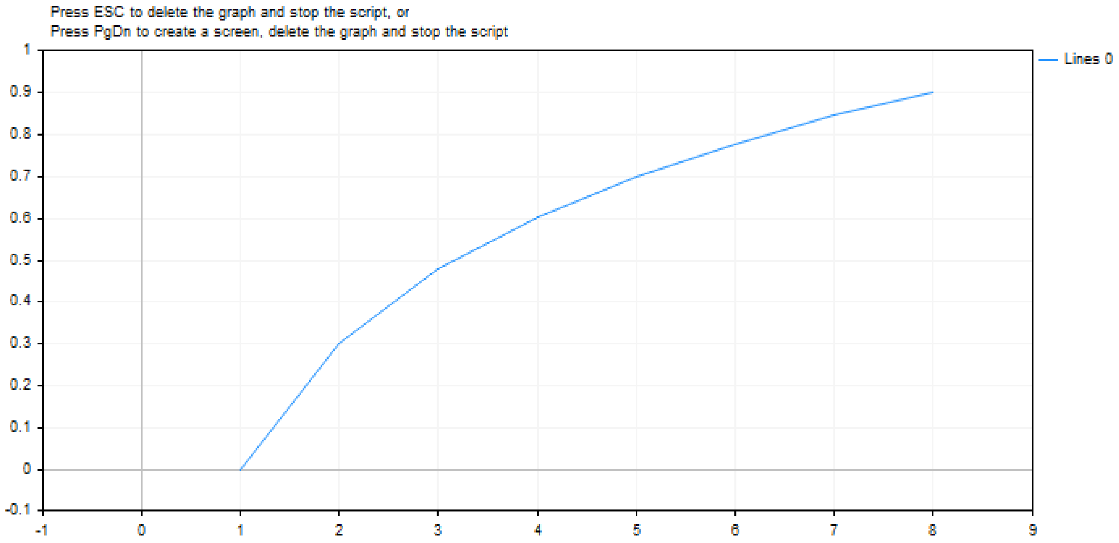
Instead of MathLog10() you can use log10().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 9 values from 0 to 8 with step 1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- calculate the decimal logarithm for each value of the X vector
   X=MathLog10(X);
   Print("MathLog10(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
   /*
   result:
   vector X =
   [0,1,2,3,4,5,6,7,8]
   MathLog10(X) =
   [-inf,0,0.3010299956639812,0.4771212547196624,0.6020599913279624,0.698970004336018
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
```

```
    ExtGraph.IndentUp(30);
    ExtGraph.CurvePlotAll();
    string text1="Press ESC to delete the graph and stop the script, or";
    string text2="Press PgDn to create a screen, delete the graph and stop the script";
    ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
    ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
    ExtGraph.Update();
   }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

Real types (double, float), Statistics, Scientific Charts, Client Terminal Properties

MathMax

The function returns the maximal value of two values.

```
double  MathMax(
   double  value1,      // first value
   double  value2       // second value
   );
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The largest of the two values.

Note

Instead of MathMax() you can use fmax(). Functions fmax(), fmin(), MathMax(), MathMin() can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically cast to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

Example:

```
//--- input parameters
input int                 InpPeriod = 10;           // Moving average calculation pe
input ENUM_MA_METHOD      InpMethod = MODE_SMA;     // Moving average calculation me
input ENUM_APPLIED_PRICE  InpPrice  = PRICE_CLOSE;  // Moving average calculation pr

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- if the moving average period is set to a value less than 1, then the default val
   int period=(InpPeriod<1 ? 10 : InpPeriod);
//--- create the Moving Average indicator handle
   int handle=iMA(Symbol(),Period(),period,0,InpMethod,InpPrice);
   if(handle==INVALID_HANDLE)
     {
      Print("Failed to create the Moving Average indicator handle. Error ",GetLastErro
      return;
     }
//--- get the current Bid price
   double bid=0;
   ResetLastError();
   if(!SymbolInfoDouble(Symbol(),SYMBOL_BID,bid))
     {
      Print("Failed to get Bid price. Error ",GetLastError());
      return;
     }
//--- get the moving average value on the current bar
   double array[1];
   int    copied=CopyBuffer(handle,0,0,1,array);
   if(copied!=1)
     {
      Print("Failed to get Moving Average data. Error ",GetLastError());
      return;
     }
//--- get the highest price of the two (Bid price and Moving Average value) and displa
   double max_price=MathMax(bid,array[0]);
   PrintFormat("Bid: %.*f, Moving Average: %.*f, highest price of the two: %.*f",_Dig
   PrintFormat("Bid price %s moving average",(bid>array[0] ? "higher" : bid<array[0]
  }
```

MathMin

The function returns the minimal value of two values.

```
double  MathMin(
   double  value1,      // first value
   double  value2       // second value
   );
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The smallest of the two values.

Note

Instead of MathMin() you can use fmin(). Functions fmax(), fmin(), MathMax(), MathMin() can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically cast to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

Example:

```
//--- input parameters
input uint  InpBars  =  100000;  // Desired number of bars

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of available bars on the server
   uint bars_total = Bars(Symbol(),Period());
   if(bars_total==0)
     {
      PrintFormat("Data for timeseries %s %s not yet generated in the terminal. Please
      return;
     }
//--- get the minimum number of bars from two values - from those available on the se
   int bars = (int)MathMin(bars_total,InpBars);
//--- if more bars are requested than are available on the server, report this in the
```

```
   if(bars_total<InpBars)
      PrintFormat("Number of bars on the server (%u) is less than requested (%u)",bars
//--- display the number of bars available for work in the journal
   Print("Bars available for work: ",bars);
  }
```

MathMod

The function returns the real remainder of division of two numbers.

```
double  MathMod(
   double  value,       // dividend value
   double  value2       // divisor value
   );
```

Parameters

value

[in] Dividend value.

value2

[in] Divisor value.

Return Value

The MathMod function calculates the real remainder f from expression val/y so that val = i * y + f , where i is an integer, f has the same sign as val, and the absolute value of f is less than the absolute value of y.

Note

Instead of MathMod() you can use fmod().

Example:

```
#property script_show_inputs

//--- input parameters
input double   InpDividentValue  =  10;    // Dividend value
input double   InpDivisorValue   =  3;     // Divisor value

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the real remainder of the division of the numbers entered in the inputs
   double res=MathMod(InpDividentValue,InpDivisorValue);
//--- print the result in the journal
   PrintFormat("Real remainder when dividing %.2f by %.2f = %.2f",InpDividentValue,Inp
  }
```

MathPow

The function raises a base to a specified power.

```
double  MathPow(
   double  base,          // base
   double  exponent       // exponent value
   );
```

Parameters

base

[in] Base.

exponent

[in] Exponent value.

Return Value

Value of base raised to the specified power.

Note

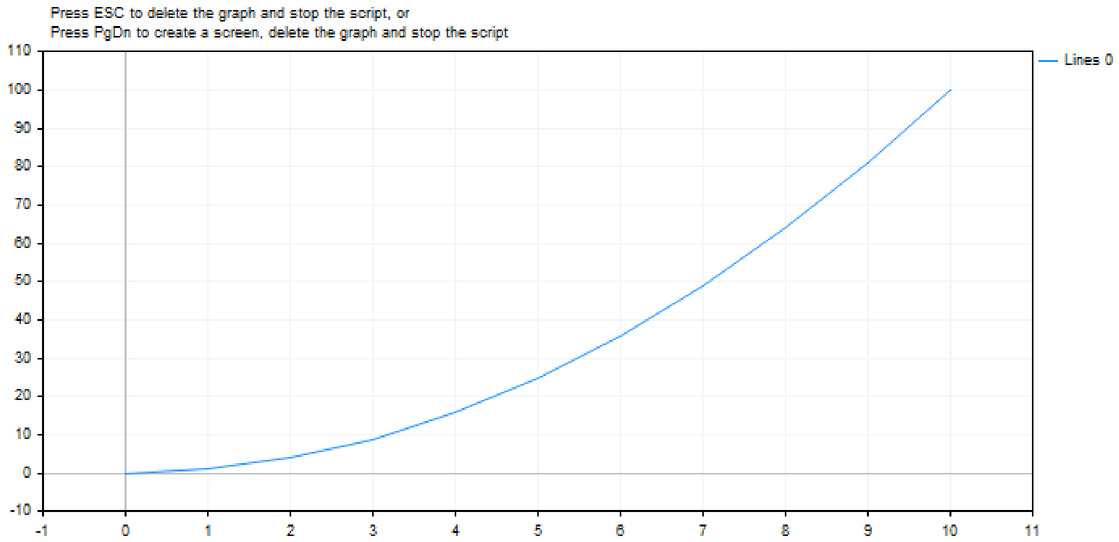
Instead of MathPow() you can use pow().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#property script_show_inputs

#include <Graphics\Graphic.mqh>

//--- input parameters
input double   InpExponentValue  =  2;    // exponent value

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 11 values from 0 to 10 with step 1
   vector X(11,VectorArange);
   Print("vector X = \n",X);
//--- calculate each value of the X vector raised to the InpExponentValue power
   X=MathPow(X,InpExponentValue);
   Print("MathPow(X,",(string)InpExponentValue,") = \n",X);

```

```
//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
   /*
   result:
   vector X = 
   [0,1,2,3,4,5,6,7,8,9,10]
   MathPow(X,2.0) = 
   [0,1,4,9,16,25,36,49,64,81,100]
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathRand

Returns a pseudorandom integer within the range of 0 to 32767.

```
int  MathRand();
```

Return Value

Integer value within the range of 0 to 32767.

Note

Before the first call of the function, it's necessary to call MathSrand to set the generator of pseudorandom numbers to the initial state.

Note

Instead of MathRand() you can use rand().

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set a new initial state to generate a series of pseudo-random integers at each
   MathSrand(GetTickCount());
//--- in a loop, display 10 generated pseudo-random integers in the journal
   for(int i=0; i<10; i++)
     {
      int rand_value=MathRand();
      PrintFormat("Pseudorandom integer №%d: %u",i+1,rand_value);
     }
  }
```

MathRound

The function returns a value rounded off to the nearest integer of the specified numeric value.

```
double  MathRound(
   double  value      // value to be rounded
   );
```

Parameters

value

[in] Numeric value before rounding.

Return Value

Value rounded till to the nearest integer.

Note

Instead of MathRound() you can use round().

Example:

```
#define VALUES_TOTAL 31

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare variables for conversion
   double value=0;                        // real number for MathRound conversion
   int    round_value=0;                  // get the result here
//--- in a loop by the number of decimal increments of a real number
   for(int i=0; i<VALUES_TOTAL; i++)
     {
      //--- increase the real number value,
      //--- get a numeric value rounded to the nearest integer
      //--- and display control values in the journal
      value+=0.1;
      round_value=(int)MathRound(NormalizeDouble(value,1));
      PrintFormat("value: %.1f, round value: %d",value,round_value);
      /*
      result:
      value: 0.1, round value: 0
      value: 0.2, round value: 0
      value: 0.3, round value: 0
      value: 0.4, round value: 0
      value: 0.5, round value: 1
      value: 0.6, round value: 1
      value: 0.7, round value: 1
      value: 0.8, round value: 1
      value: 0.9, round value: 1
      value: 1.0, round value: 1
      value: 1.1, round value: 1
      value: 1.2, round value: 1
      value: 1.3, round value: 1
      value: 1.4, round value: 1
      value: 1.5, round value: 2
      value: 1.6, round value: 2
      value: 1.7, round value: 2
      value: 1.8, round value: 2
      value: 1.9, round value: 2
      value: 2.0, round value: 2
      value: 2.1, round value: 2
      value: 2.2, round value: 2
      value: 2.3, round value: 2
      value: 2.4, round value: 2
      value: 2.5, round value: 3
      value: 2.6, round value: 3
      value: 2.7, round value: 3
      value: 2.8, round value: 3
      value: 2.9, round value: 3
      value: 3.0, round value: 3
      value: 3.1, round value: 3
      */
     }
  }
```

MathSin

Returns the sine of a specified angle.

```
double  MathSin(
   double  value      // argument in radians
   );
```

Parameters

value

[in] Angle in radians.

Return Value

Sine of an angle measured in radians. Returns value within the range of -1 to 1.

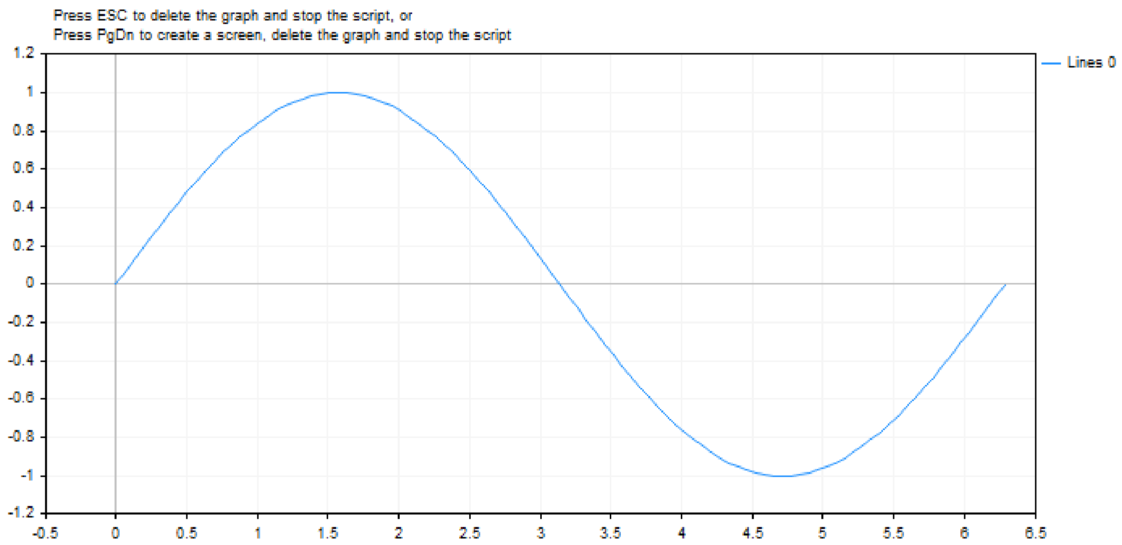
Note

Instead of MathSin() you can use sin().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the sine value for each value of the X vector
   vector Y=MathSin(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



Press ESC to delete the graph and stop the script, or
Press PgDn to create a screen, delete the graph and stop the script

See also

Real types (double, float), Statistics, Scientific Charts, Client Terminal Properties

MathSqrt

Returns the square root of a number.

```
double  MathSqrt(
   double  value      // positive number
   );
```

Parameters

value

[in] Positive numeric value.

Return Value

Square root of value. If value is negative, MathSqrt returns NaN (indeterminate value).

Note

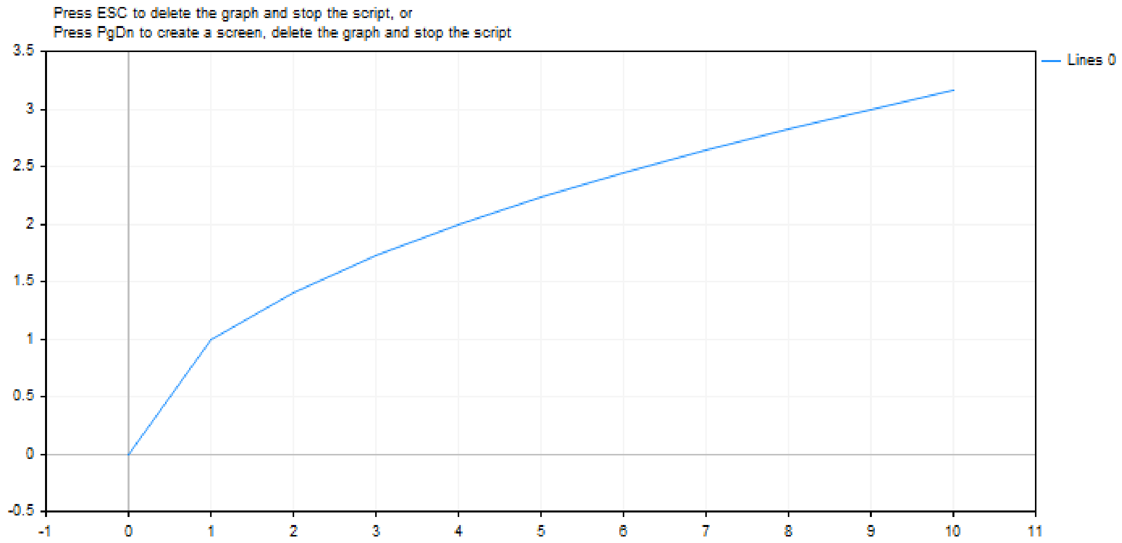
Instead of MathSqrt() you can use sqrt().

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 11 values from 0 to 10 with step 1
   vector X(11,VectorArange);
   Print("vector X = \n",X);
//--- calculate the square root of each value of the X vector
   X=MathSqrt(X);
   Print("MathSqrt(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
```

```
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
   /*
   result:
   vector X = 
   [0,1,2,3,4,5,6,7,8,9,10]
   MathSqrt(X) = 
   [0,1,1.414213562373095,1.732050807568877,2,2.23606797749979,2.449489742783178,2.64
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

Real types (double, float), Statistics, Scientific Charts, Client Terminal Properties

MathSrand

Sets the starting point for generating a series of pseudorandom integers.

```
void  MathSrand(
   int   seed      // initializing number
   );
```

Parameters

seed

[in] Starting number for the sequence of random numbers.

Return Value

No return value.

Note

The MathRand() function is used for generating a sequence of pseudorandom numbers. Call of MathSrand() with a certain initializing number allows to always produces the same sequence of pseudorandom numbers.

To ensure receipt of non-recurring sequence, use the call of MathSrand(GetTickCount()), since the value of GetTickCount() increases from the moment of the start of the operating system and is not repeated within 49 days, until the built-in counter of milliseconds overflows. Use of MathSrand(TimeCurrent()) is not suitable, because the TimeCurrent() function returns the time of the last tick, which can be unchanged for a long time, for example at the weekend.

Initialization of the random number generator using MathSrand() for indicators and Expert Advisors is better performed in the OnInit() handler; it saves you from the following multiple restarts of the generator in OnTick() and OnCalculate().

Instead of the MathSrand() function you can use the srand() function.

Example:

```
#property description "The indicator shows the central limit theorem, which states:"
#property description "The sum of a sufficiently large number of weakly dependent rand
#property description "having approximately equal magnitude (none of the summands domi
#property description "or makes a determining contribution to the sum), has a distribu

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- Properties of the graphical construction
#property indicator_label1  "Label"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRoyalBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  5
//--- An input variable
input int      sample_number=10;
//--- An indicator buffer to for drawing the distribution
```

```
double         LabelBuffer[];
//--- A counter of ticks
double         ticks_counter;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- turn the indicator buffer around from the present to the past
   ArraySetAsSeries(LabelBuffer,true);
//--- Initialize the generator of random numbers
   MathSrand(GetTickCount());
//--- Initialize the counter of ticks
   ticks_counter=0;
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- For a zero counter reset the indicator buffer
   if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- Increase the counter
   ticks_counter++;
//--- We should periodically reset the counter ticks, to revive the distribution
   if(ticks_counter>100)
     {
      Print("We've reset the indicator values, let's start filling the cells once aga
      ticks_counter=0;
     }
//--- Get a sample of random values as the sum of three numbers from 0 to 7
   for(int i=0;i<sample_number;i++)
     {
      //--- Calculation of the index of the cell, where the random number falls as the
      int rand_index=0;
      //--- Get three random numbers from 0 to 7
      for(int k=0;k<3;k++)
        {
          //--- A remainder in the division by 7 will return a value from 0 to 6
          rand_index+=MathRand()%7;
         }
       //--- Increase the value in the cell number rand_index by 1
       LabelBuffer[rand_index]++;
      }
//--- Exit the OnCalculate() handler
   return(rates_total);
  }
```

MathTan

The function returns a tangent of a number.

```
double  MathTan(
   double  rad      // argument in radians
   );
```

Parameters

rad

[in] Angle in radians.

Return Value

Tangent of rad. If rad is greater than or equal to 263, or less than or equal to -263, a loss of significance in the result occurs, in which case the function returns an indefinite number.

Note

Instead of MathTan() you can use tan().

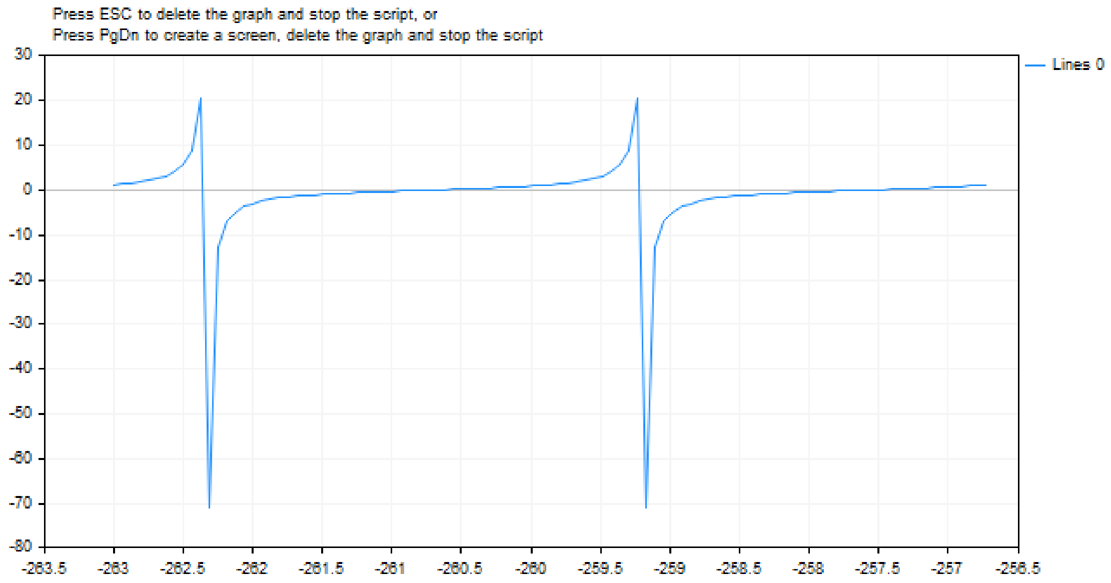
Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=-263;
//--- get 101 values from -263 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the tangent value for each value of the X vector
   vector Y=MathTan(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);
```

```
//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
```

```
 int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
 if(selected<1)
   {
    if(selected<0)
       PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
    return false;
   }

 bool res=false;
 if(ChartSetInteger(0,CHART_SHOW,false))
    res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
 ChartSetInteger(0,CHART_SHOW,true);
 return(res);
}
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathIsValidNumber

It checks the correctness of a real number.

```
bool  MathIsValidNumber(
   double  number      // number to check
   );
```

Parameters

number

[in] Checked numeric value.

Return Value

It returns true, if the checked value is an acceptable real number. If the checked value is a plus or minus infinity, or "not a number" (NaN), the function returns false.

Example:

```
   double abnormal=MathArcsin(2.0);
   if(!MathIsValidNumber(abnormal)) Print("Attention! MathArcsin(2.0) = ",abnormal);
```

See also

[Real types (double, float)]

MathExpm1

Returns the value of the expression MathExp(x)-1.

```
double  MathExpm1(
   double  value       // power for the number e
   );
```

Parameters

value

[in] The number specifying the power.

Return Value

A value of the double type. In case of overflow the function returns INF (infinity), in case of underflow MathExpm1 returns 0.

Note

At values of *x* close to 0, the MathExpm1(x) function generates much more accurate values than the MathExp(x)-1 function.

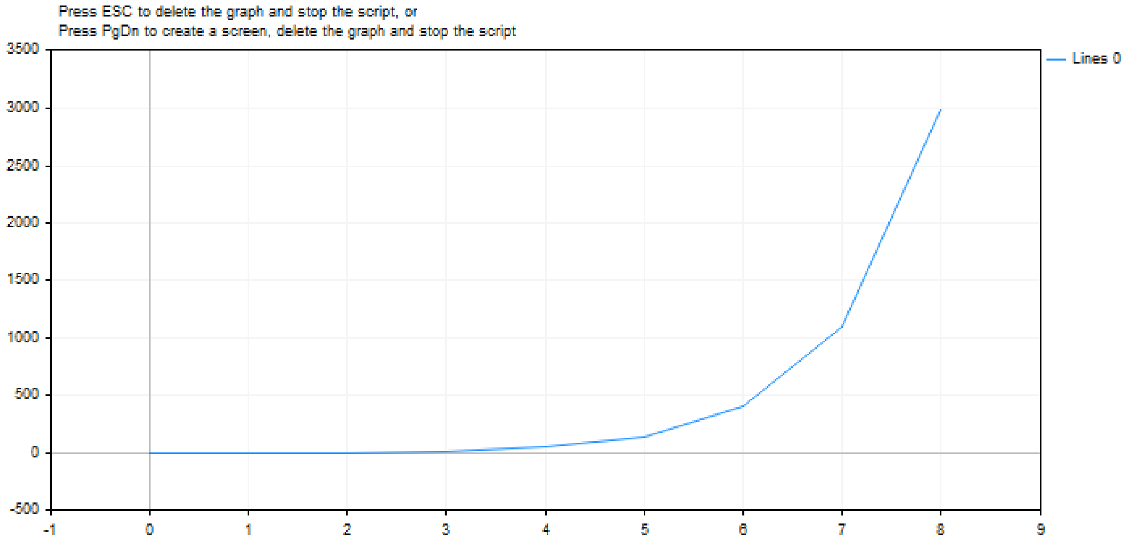
Instead of the MathExpm1() function you can use the expm1() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 9 values from 0 to 8 with step 1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- calculate the ("e"(Euler number) to the power of each X vector value) - 1
   X=MathExpm1(X);
   Print("MathExpm1(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
```

```
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathLog1p

Returns the value of the expression MathLog(1+x).

```
double  MathLog1p(
   double  value      // value to take the logarithm
   );
```

Parameters

value

[in] The value, the logarithm of which is to be calculated.

Return Value

The natural logarithm of the value (value + 1) if successful. If value is < -1, the function returns NaN (undefined value). If value is equal to -1, the function returns INF (infinity).

Note

At values of x close to 0, the MathLog1p(x) function generates much more accurate values than the MathLog(1+x) function.

Instead of the MathLog1p() function you can use the log1p() function.

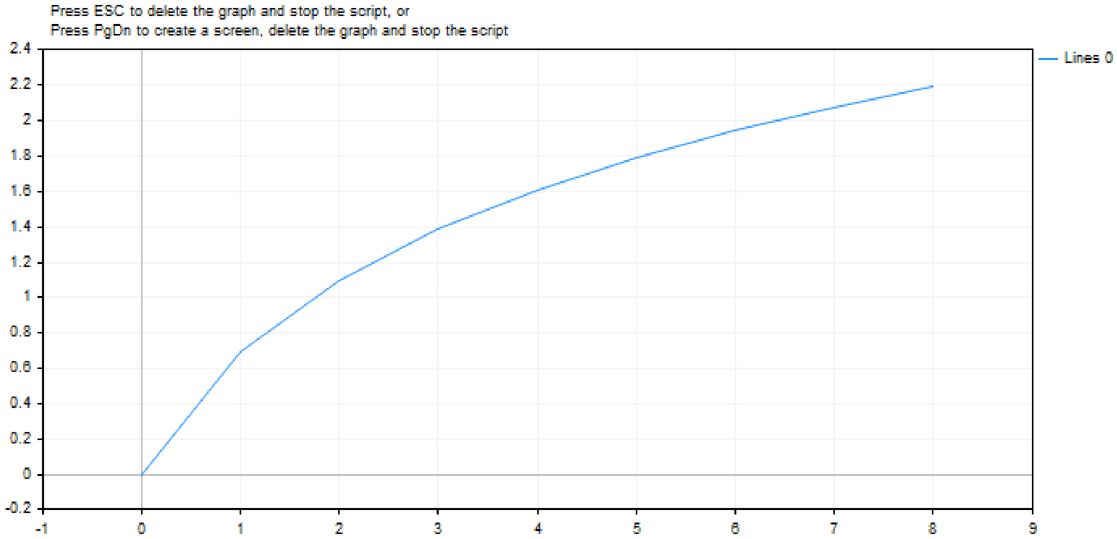
Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get 9 values from 0 to 8 with step 1
   vector X(9,VectorArange);
   Print("vector X = \n",X);
//--- for each vector value, calculate the logarithm for the expression (1 + X vector
   X=MathLog1p(X);
   Print("MathLog1p(X) = \n",X);

//--- transfer the calculated values from the vector to the array
   double y_array[];
   X.Swap(y_array);

//--- draw a graph of the calculated vector values
```

```
   CurvePlot(y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
   /*
   result:
   vector X = 
   [0,1,2,3,4,5,6,7,8]
   MathLog1p(X) = 
   [0,0.6931471805599453,1.09861228866811,1.386294361119891,1.6094379124341,1.7917594
   */
  }
//+------------------------------------------------------------------+
//| Fill a vector with 'value' in 'step' increments                  |
//+------------------------------------------------------------------+
template<typename T>
void VectorArange(vector<T> &vec,T value=0.0,T step=1.0)
  {
   for(ulong i=0; i<vec.Size(); i++,value+=step)
      vec[i]=value;
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
```

```
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArccosh

Returns the hyperbolic arccosine.

```
double  MathArccosh(
   double  value      // 1 <= value < ∞
   );
```

Parameters

value

[in] The value, the hyperbolic arccosine of which is to be calculated.

Return Value

The hyperbolic arccosine of the number. If value is less than +1, the function returns NaN (undefined value).

Note

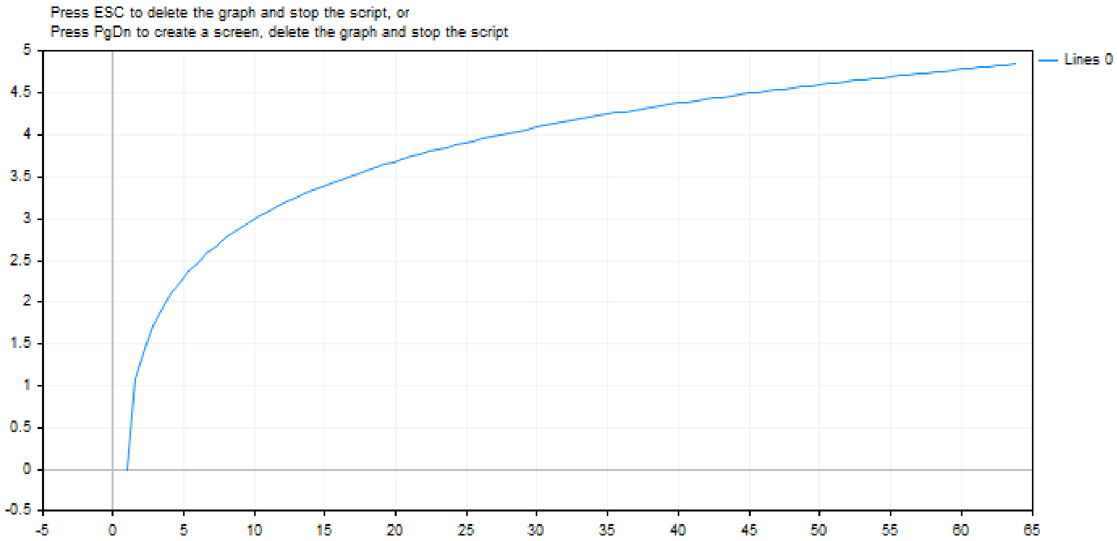
Instead of the MathArccosh() function you can use the acosh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/10);
   delta[0]=1;
//--- get 101 values from 1 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic arc cosine value for each value of the X vector
   vector Y=MathArccosh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

```

```
//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and saves the image to a file                  |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



Press ESC to delete the graph and stop the script, or
Press PgDn to create a screen, delete the graph and stop the script

See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArcsinh

Returns the hyperbolic arcsine.

```
double  MathArcsinh(
   double  value       // -∞ < value < +∞
   );
```

Parameters

val

[in] The value, the hyperbolic arcsine of which is to be calculated.

Return Value

The hyperbolic arcsine of the number.

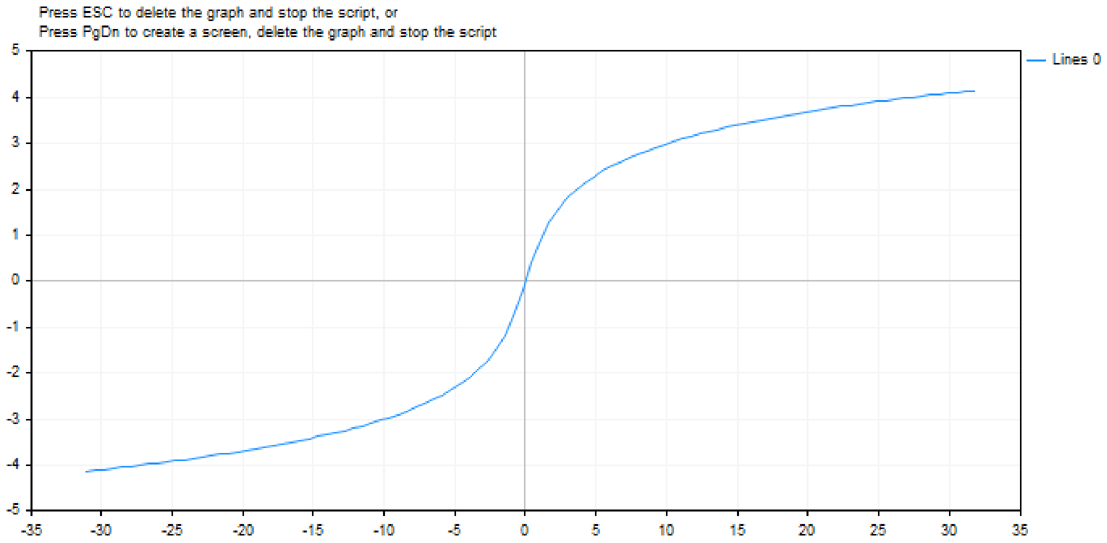
Note

Instead of the MathArcsinh() function you can use the asinh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/10);
   delta[0]=-31;
//--- get 101 values from -31 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic arc sine value for each value of the X vector
   vector Y=MathArcsinh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
 if(selected<1)
   {
    if(selected<0)
       PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
    return false;
   }

 bool res=false;
 if(ChartSetInteger(0,CHART_SHOW,false))
    res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
 ChartSetInteger(0,CHART_SHOW,true);
 return(res);
}
```

Result:



See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathArctanh

Returns the hyperbolic arctangent.

```
double  MathArctanh(
   double  value      // value in the range of -1 < value < 1
   );
```

Parameters

value

[in] Number within the range of -1 < value < 1, which represents the tangent.

Return Value

The hyperbolic arctangent of the number.

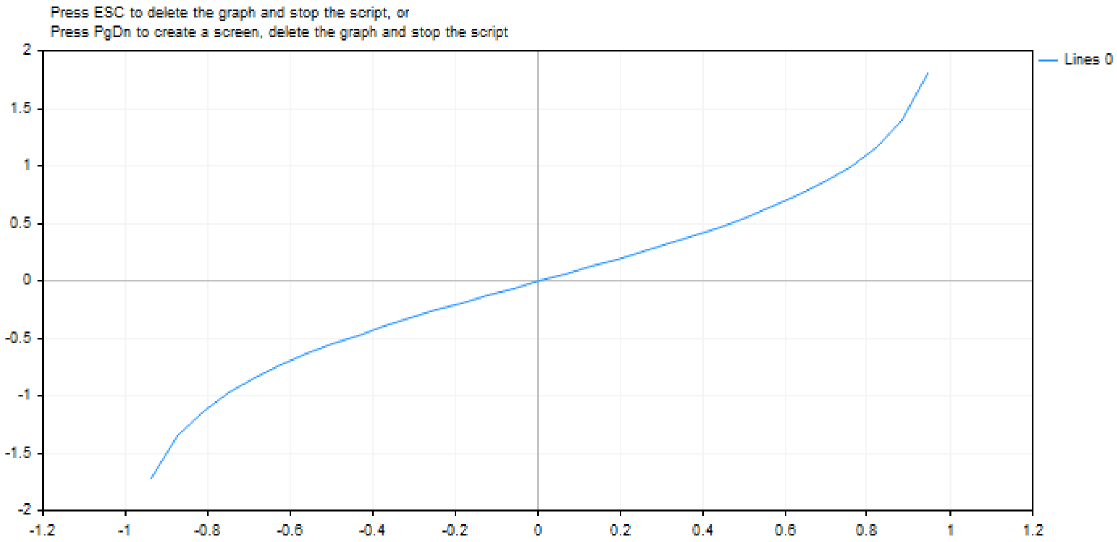
Note

Instead of the MathArctanh() function you can use the atanh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(34,2*M_PI/100);
   delta[0]=-1;
//--- get 34 values from -1 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic arc tangent value for each value of the X vector
   vector Y=MathArctanh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathCosh

Returns the hyperbolic cosine of the number.

```
double  MathCosh(
   double  value      // number
   );
```

Parameters

value

[in] Value.

Return Value

The hyperbolic cosine of the number, value within the range of +1 to positive infinity.

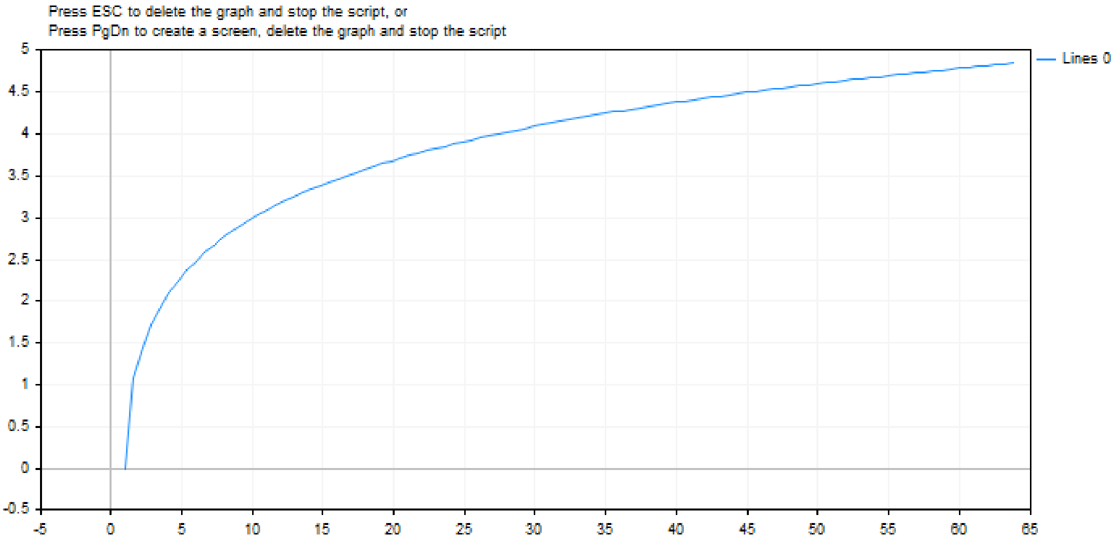
Note

Instead of the MathCosh() function you can use the cosh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic cosine value for each value of the X vector
   vector Y=MathCosh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQLI
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRIT
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



See also

[Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathSinh

Returns the hyperbolic sine of the number.

```
double  MathSinh(
   double  value      // number
   );
```

Parameters

value

[in] Value.

Return Value

The hyperbolic sine of the number.

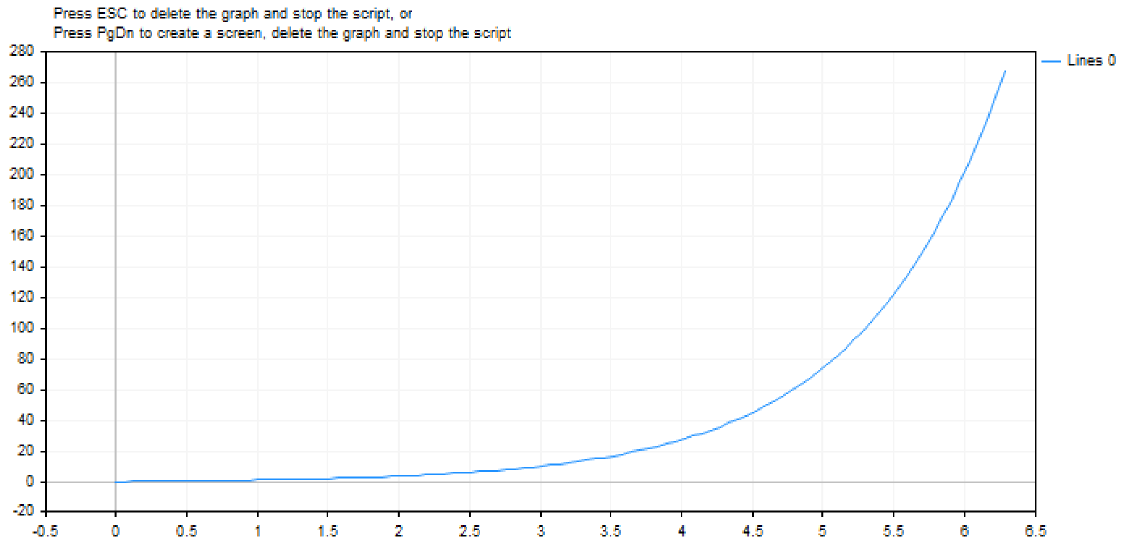
Note

Instead of the MathSinh() function you can use the sinh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic sine value for each value of the X vector
   vector Y=MathSinh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
 if(selected<1)
   {
    if(selected<0)
       PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION_
    return false;
   }

 bool res=false;
 if(ChartSetInteger(0,CHART_SHOW,false))
    res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
 ChartSetInteger(0,CHART_SHOW,true);
 return(res);
}
```

Result:



Press ESC to delete the graph and stop the script, or
Press PgDn to create a screen, delete the graph and stop the script

See also

[Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathTanh

Returns the hyperbolic tangent of the number.

```
double  MathTanh(
   double  value      // number
   );
```

Parameters

value

[in] Value.

Return Value

The hyperbolic tangent of the number, value within the range of -1 to +1.

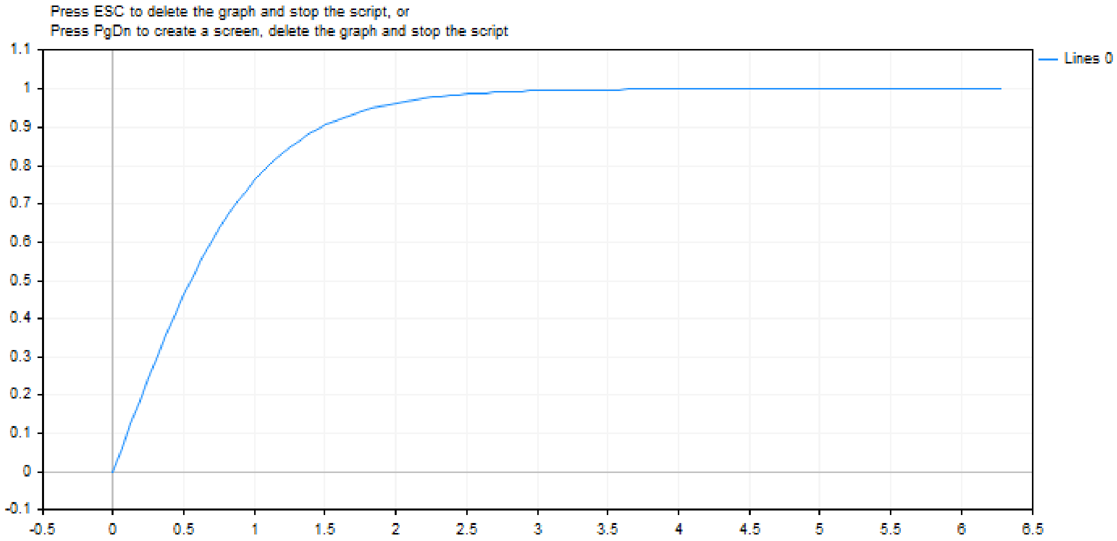
Note

Instead of the MathTanh() function you can use the tanh() function.

Example:

```
#define GRAPH_WIDTH  750
#define GRAPH_HEIGHT 350

#include <Graphics\Graphic.mqh>

CGraphic ExtGraph;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   vector delta=vector::Full(101,2*M_PI/100);
   delta[0]=0;
//--- get 101 values from 0 to 2 pi with delta step
   vector X=delta.CumSum();
//--- calculate the hyperbolic tangent value for each value of the X vector
   vector Y=MathTanh(X);

//--- transfer the calculated values from vectors to arrays
   double x_array[],y_array[];
   X.Swap(x_array);
   Y.Swap(y_array);

//--- draw a graph of the calculated vector values
   CurvePlot(x_array,y_array,clrDodgerBlue);

//--- wait for pressing the Escape or PgDn keys to delete the graph (take a screenshot
   while(!IsStopped())
     {
      if(StopKeyPressed())
         break;
      Sleep(16);
     }

//--- clean up
   ExtGraph.Destroy();
  }
//+------------------------------------------------------------------+
//| When pressing ESC, return 'true'                                 |
//| When pressing PgDn, take a graph screenshot and return 'true'    |
//| Otherwise, return 'false'                                        |
//+------------------------------------------------------------------+
bool StopKeyPressed()
  {
//--- if ESC is pressed, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)!=0)
      return(true);
//--- if PgDn is pressed and a graph screenshot is successfully taken, return 'true'
   if(TerminalInfoInteger(TERMINAL_KEYSTATE_PAGEDOWN)!=0 && MakeAndSaveScreenshot(MQL
      return(true);
//--- return 'false'
   return(false);
  }
//+------------------------------------------------------------------+
//| Create a graph object and draw a curve                           |
//+------------------------------------------------------------------+
void CurvePlot(double &x_array[], double &y_array[], const color colour)
  {
   ExtGraph.Create(ChartID(), "Graphic", 0, 0, 0, GRAPH_WIDTH, GRAPH_HEIGHT);
   ExtGraph.CurveAdd(x_array, y_array, ColorToARGB(colour), CURVE_LINES);
   ExtGraph.IndentUp(30);
   ExtGraph.CurvePlotAll();
   string text1="Press ESC to delete the graph and stop the script, or";
   string text2="Press PgDn to create a screen, delete the graph and stop the script";
   ExtGraph.TextAdd(54, 9, text1, ColorToARGB(clrBlack));
   ExtGraph.TextAdd(54,21, text2, ColorToARGB(clrBlack));
   ExtGraph.Update();
  }
//+------------------------------------------------------------------+
//| Take a screenshot and save the image to a file                   |
//+------------------------------------------------------------------+
bool MakeAndSaveScreenshot(const string file_name)
  {
   string file_names[];
   ResetLastError();
   int selected=FileSelectDialog("Save Picture", NULL, "All files (*.*)|*.*", FSD_WRI
   if(selected<1)
     {
      if(selected<0)
         PrintFormat("%s: FileSelectDialog() function returned error %d", __FUNCTION__
      return false;
     }

   bool res=false;
   if(ChartSetInteger(0,CHART_SHOW,false))
      res=ChartScreenShot(0, file_names[0], GRAPH_WIDTH, GRAPH_HEIGHT);
   ChartSetInteger(0,CHART_SHOW,true);
   return(res);
  }
```

Result:



Press ESC to delete the graph and stop the script, or
Press PgDn to create a screen, delete the graph and stop the script

See also

[Real types (double, float)](), [Statistics](), [Scientific Charts](), [Client Terminal Properties]()

MathSwap

Change the order of bytes in the ushort type value.

```
ushort  MathSwap(
   ushort  value       // value
   );
```

Parameters

value

[in] Value for changing the order of bytes.

Return Value

ushort value with the reverse byte order.

MathSwap

Change the order of bytes in the uint type value.

```
uint  MathSwap(
   uint    value       // value
   );
```

Parameters

value

[in] Value for changing the order of bytes.

Return Value

uint value with the reverse byte order.

MathSwap

Change the order of bytes in the ulong type value.

```
ulong  MathSwap(
   ulong   value       // value
   );
```

Parameters

value

[in] Value for changing the order of bytes.

Return Value

ulong value with the reverse byte order.

Example:

```
#property script_show_inputs

input ulong  InpLongValue  =  1;    // Enter any ulong value here
input uint   InpIntValue   =  2;    // Enter any uint value here
input ushort InpShortValue =  3;    // Enter any ushort value here

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {

//--- print the values entered and converted by MathSwap() in decimal and binary repre
   Print(ValueDescription(InpLongValue));
   Print(ValueDescription(InpIntValue));
   Print(ValueDescription(InpShortValue));
   /*
   result:
   ulong value: 1
   ulong value: 72057594037927936 using MathSwap()
   binary ulong value: 00000000000000000000000000000000000000000000000000000000000000
   binary ulong value: 00000001000000000000000000000000000000000000000000000000000000

   uint value: 2
   uint value: 33554432 using MathSwap()
   binary uint value: 00000000000000000000000000000010
   binary uint value: 00000010000000000000000000000000 using MathSwap()

   ushort value: 3
   ushort value: 768 using MathSwap()
   binary ushort value: 0000000000000011
   binary ushort value: 0000001100000000 using MathSwap()
   */
  }
//+------------------------------------------------------------------+
//| Return the text describing the variable values                   |
//+------------------------------------------------------------------+
template <typename T>
string ValueDescription(T x)
  {
   int    num_bits   = sizeof(T)*8;
   string type_name  = typename(T);
   string bin_x      = NumberToBinaryString(x);
   string bin_swap_x = NumberToBinaryString(MathSwap(x));
   return(StringFormat("%s value: %lld\n%s value: %lld using MathSwap()\nbinary %s va
                       type_name, x, type_name, MathSwap(x), type_name, num_bits, bin_
  }
//+------------------------------------------------------------------+
//| Return the binary representation of a number as a string         |
//+------------------------------------------------------------------+
template <typename T>
string NumberToBinaryString(T x)
  {
   string res  = "";
   int    i    = -1;
   uchar  size = sizeof(T)*8-1;
   ulong  mask = (ulong)1<<size;
   while(!((x<<++i)& mask));
   for(; i <=size; i++)
      res += !((x<<i)& mask) ? "0" : "1";
   return res;
  }
```

See also

Network functions, SocketRead, SocketSend, SocketTlsRead, SocketTlsReadAvailable, SocketTlsSend

String Functions

This is a group of functions intended for working with data of the string type.

Function	Action
StringAdd	Adds a string to the end of another string
StringBufferLen	Returns the size of buffer allocated for the string
StringCompare	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1 (minus 1) - if the first string is less than the second one
StringConcatenate	Forms a string of parameters passed
StringFill	Fills out a specified string by selected symbols
StringFind	Search for a substring in a string
StringGetCharacter	Returns the value of a number located in the specified string position
StringInit	Initializes string by specified symbols and provides the specified string length
StringLen	Returns the number of symbols in a string
StringSetLength	Sets a specified length (in characters) for a string
StringReplace	Replaces all the found substrings of a string by a set sequence of symbols
StringReserve	Reserves the buffer of a specified size for a string in memory.
StringSetCharacter	Returns a copy of a string with a changed value of a symbol in a specified position
StringSplit	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained
StringSubstr	Extracts a substring from a text string starting from a specified position
StringToLower	Transforms all symbols of a selected string to lowercase
StringToUpper	Transforms all symbols of a selected string into capitals
StringTrimLeft	Cuts line feed characters, spaces and tabs in the left part of the string
StringTrimRight	Cuts line feed characters, spaces and tabs in the right part of the string

StringAdd

The function adds a substring to the end of a string.

```
bool  StringAdd(
   string&  string_var,         // string, to which we add
   string   add_substring       // string, which is added
   );
```

Parameters

string_var

[in][out] String, to which another one is added.

add_substring

[in] String that is added to the end of a source string.

Return Value

In case of success returns true, otherwise false. In order to get an error code, the GetLastError() function should be called.

Example:

```
void OnStart()
  {
   long length=1000000;
   string a="a",b="b",c;
//--- first method
   uint start=GetTickCount(),stop;
   long i;
   for(i=0;i<length;i++)
     {
      c=a+b;
     }
   stop=GetTickCount();
   Print("time for 'c = a + b' = ",(stop-start)," milliseconds, i = ",i);

//--- second method
   start=GetTickCount();
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringAdd(a,b)' = ",(stop-start)," milliseconds, i = ",i);

//--- third method
   start=GetTickCount();
   a="a"; // re-initialize variable a
   for(i=0;i<length;i++)
```

```
     {
      StringConcatenate(c,a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringConcatenate(c,a,b)' = ",(stop-start)," milliseconds, i = ",
  }
```

See also

[StringConcatenate](), [StringSplit](), [StringSubstr]()

StringBufferLen

The function returns the size of buffer allocated for the string.

```
int  StringBufferLen(
   string  string_var      // string
   )
```

Parameters

string_var

[in] String.

Return Value

The value 0 means that the string is constant and buffer size can't be changed. -1 means that the string belongs to the client terminal, and modification of the buffer contents can have indeterminate results.

Example:

```
void OnStart()
  {
   long length=1000;
   string a="a",b="b";
//---
   long i;
   Print("before: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   Print("after: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
  }
```

See also

StringAdd, StringInit, StringLen, StringFill

StringCompare

The function compares two strings and returns the comparison result in form of an integer.

```
int  StringCompare(
   const string&  string1,                  // the first string in the comparison
   const string&  string2,                  // the second string in the comparison
   bool           case_sensitive=true       // case sensitivity mode selection for the
   );
```

Parameters

string1

[in] The first string.

string2

[in] The second string.

case_sensitive=true

[in] Case sensitivity mode selection. If it is true, then "A">"a". If it is false, then "A"="a". By default the value is equal to true.

Return Value

- -1 (minus one), if string1<string2
- 0 (zero), if string1=string2
- 1 (one), if string1>string2

Note

The strings are compared symbol by symbol, the symbols are compared in the alphabetic order in accordance with the current code page.

Example:

```
void OnStart()
  {
//--- what is larger - apple or home?
   string s1="Apple";
   string s2="home";

//--- compare case sensitive
   int result1=StringCompare(s1,s2);
   if(result1>0) PrintFormat("Case sensitive comparison: %s > %s",s1,s2);
   else
     {
      if(result1<0)PrintFormat("Case sensitive comparison: %s < %s",s1,s2);
      else PrintFormat("Case sensitive comparison: %s = %s",s1,s2);
     }

//--- compare case-insensitive
   int result2=StringCompare(s1,s2,false);
   if(result2>0) PrintFormat("Case insensitive comparison: %s > %s",s1,s2);
   else
```

```
      {
       if(result2<0)PrintFormat("Case insensitive comparison: %s < %s",s1,s2);
       else PrintFormat("Case insensitive comparison: %s = %s",s1,s2);
      }
/* Result
     Case-sensitive comparison: Apple < home
     Case insensitive comparison: Apple < home
*/
  }
```

See also

String Type, CharToString(), ShortToString(), StringToCharArray(), StringToShortArray(), StringGetCharacter(), Use of a Codepage

StringConcatenate

The function forms a string of passed parameters and returns the size of the formed string. Parameters can be of any type. Number of parameters can't be less than 2 or more than 64.

```
int  StringConcatenate(
   string&  string_var,   // string to form
   void argument1         // first parameter of any simple type
   void argument2         // second parameter of any simple type
   ...                    // next parameter of any simple type
   );
```

Parameters

string_var

[out] String that will be formed as a result of concatenation.

argumentN

[in] Any comma separated values. From 2 to 63 parameters of any simple type.

Return Value

Returns the string length, formed by concatenation of parameters transformed into string type. Parameters are transformed into strings according to the same rules as in Print() and Comment().

Example:

```
void OnStart()
  {
//--- declare and define variables participating in concatenation
   string text="";
   string text1="This script shows how the StringConcatenate() function works.\n";
   string text2="This is the second line, at the end of which there is a line break c
   string text3="This is line number ";
   int    num3=3;
   string text31=", the number of which is entered into the function as a separate pa
   string textN="\n";
   string text4="This is line number 4, preceded by a separate parameter with a line
   int    length=StringConcatenate(text, text1, text2, text3, num3, text31, textN, te
   Print(text, "\nLength of the resulting string = ", length);

   /*
   Result
   This script shows how the StringConcatenate() function works.
   This is the second line, at the end of which there is a line break control code.
   This is line number 3, the number of which is entered into the function as a separ
   This is line number 4, preceded by a separate parameter with a line break code.
   Line 5 includes a real number: 0.12345
   Length of the resulting string = 358
   */
  }
```

See also

StringAdd, StringSplit, StringSubstr

StringFill

It fills out a selected string by specified symbols.

```
bool  StringFill(
   string&    string_var,       // string to fill
   ushort     character         // symbol that will fill the string
   );
```

Parameters

string_var

[in][out] String, that will be filled out by the selected symbol.

character

[in] Symbol, by which the string will be filled out.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Note

Filling out a string at place means that symbols are inserted directly to the string without transitional operations of new string creation or copying. This allows to save the operation time.

Example:

```
void OnStart()
  {
   string str;
   StringInit(str,20,'_');
   Print("str = ",str);
   StringFill(str,0);
   Print("str = ",str,": StringBufferLen(str) = ", StringBufferLen(str));
  }
// Result
//   str = ____________________
//   str =  : StringBufferLen(str) = 20
//
```

See also

StringBufferLen, StringLen, StringInit

StringFind

Search for a substring in a string.

```
int  StringFind(
   string  string_value,        // string in which search is made
   string  match_substring,     // what is searched
   int     start_pos=0          // from what position search starts
   );
```

Parameters

string_value

[in] String, in which search is made.

match_substring

[in] Searched substring.

start_pos=0

[in] Position in the string from which search is started.

Return Value

Returns position number in a string, from which the searched substring starts, or -1, if the substring is not found.

Example:

```
#define   RESERVE    100

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the symbol base currency and the profit currency
   string symbol_currency_base  =SymbolInfoString(Symbol(), SYMBOL_CURRENCY_BASE);
   string symbol_currency_profit=SymbolInfoString(Symbol(), SYMBOL_CURRENCY_PROFIT);
   PrintFormat("Symbol Currency Base: %s\nSymbol Currency Profit: %s", symbol_currency

//--- in the loop through all symbols available on the server
   int total=SymbolsTotal(false), pos=-1;
   for(int i=0; i<total; i++)
     {
      //--- get the name of the next symbol
      string name=SymbolName(i, false);

      //--- look for a substring in the symbol name with the name of the base currency
      //--- if a substring is found, display the symbol name, its index in the currenc
      pos = StringFind(name, symbol_currency_base);
      if(pos >= 0)
      PrintFormat("The '%s' symbol at index %u in the list contains the '%s' curren

   //--- look for a substring in the symbol name with the name of the quoted curren
   //--- if a substring is found, display the symbol name, its index in the currenc
   pos = StringFind(name, symbol_currency_profit);
   if(pos >= 0)
      PrintFormat("The '%s' symbol at index %u in the list contains the '%s' curren
  }

/*
Result
StringFind (EURUSD,D1)   Symbol Currency Base: EUR
StringFind (EURUSD,D1)   Symbol Currency Profit: USD
The 'EURUSD' symbol at index 0 in the list contains the 'EUR' currency. Substring p
The 'EURUSD' symbol at index 0 in the list contains the 'USD' currency. Substring p
The 'GBPUSD' symbol at index 1 in the list contains the 'USD' currency. Substring p
The 'USDCHF' symbol at index 2 in the list contains the 'USD' currency. Substring p
The 'USDJPY' symbol at index 3 in the list contains the 'USD' currency. Substring p
The 'USDCNH' symbol at index 4 in the list contains the 'USD' currency. Substring p
The 'USDRUB' symbol at index 5 in the list contains the 'USD' currency. Substring p
The 'AUDUSD' symbol at index 6 in the list contains the 'USD' currency. Substring p
The 'NZDUSD' symbol at index 7 in the list contains the 'USD' currency. Substring p
The 'USDCAD' symbol at index 8 in the list contains the 'USD' currency. Substring p
The 'USDSEK' symbol at index 9 in the list contains the 'USD' currency. Substring p
The 'USDHKD' symbol at index 10 in the list contains the 'USD' currency. Substring
The 'USDSGD' symbol at index 11 in the list contains the 'USD' currency. Substring
The 'USDNOK' symbol at index 12 in the list contains the 'USD' currency. Substring
The 'USDDKK' symbol at index 13 in the list contains the 'USD' currency. Substring
The 'USDTRY' symbol at index 14 in the list contains the 'USD' currency. Substring
The 'USDZAR' symbol at index 15 in the list contains the 'USD' currency. Substring
The 'USDCZK' symbol at index 16 in the list contains the 'USD' currency. Substring
The 'USDHUF' symbol at index 17 in the list contains the 'USD' currency. Substring
The 'USDPLN' symbol at index 18 in the list contains the 'USD' currency. Substring
The 'USDRUR' symbol at index 19 in the list contains the 'USD' currency. Substring
The 'EURAUD' symbol at index 27 in the list contains the 'EUR' currency. Substring
The 'EURCAD' symbol at index 28 in the list contains the 'EUR' currency. Substring
The 'EURCHF' symbol at index 29 in the list contains the 'EUR' currency. Substring
The 'EURCZK' symbol at index 30 in the list contains the 'EUR' currency. Substring
The 'EURDKK' symbol at index 31 in the list contains the 'EUR' currency. Substring
The 'EURGBP' symbol at index 32 in the list contains the 'EUR' currency. Substring
The 'EURHKD' symbol at index 33 in the list contains the 'EUR' currency. Substring
The 'EURHUF' symbol at index 34 in the list contains the 'EUR' currency. Substring
The 'EURJPY' symbol at index 35 in the list contains the 'EUR' currency. Substring
The 'EURNOK' symbol at index 36 in the list contains the 'EUR' currency. Substring
The 'EURNZD' symbol at index 37 in the list contains the 'EUR' currency. Substring
The 'EURPLN' symbol at index 38 in the list contains the 'EUR' currency. Substring
The 'EURRUR' symbol at index 39 in the list contains the 'EUR' currency. Substring
The 'EURRUB' symbol at index 40 in the list contains the 'EUR' currency. Substring
The 'EURSEK' symbol at index 41 in the list contains the 'EUR' currency. Substring
```

```
 The 'EURTRY' symbol at index 42 in the list contains the 'EUR' currency. Substring
 The 'EURZAR' symbol at index 43 in the list contains the 'EUR' currency. Substring
 The 'XAUUSD' symbol at index 47 in the list contains the 'USD' currency. Substring
 The 'XAUEUR' symbol at index 48 in the list contains the 'EUR' currency. Substring
 The 'XAGUSD' symbol at index 50 in the list contains the 'USD' currency. Substring
 The 'XAGEUR' symbol at index 51 in the list contains the 'EUR' currency. Substring
 The 'USDCRE' symbol at index 53 in the list contains the 'USD' currency. Substring
 The 'XPDUSD' symbol at index 65 in the list contains the 'USD' currency. Substring
 The 'XPTUSD' symbol at index 66 in the list contains the 'USD' currency. Substring
 The 'USDGEL' symbol at index 67 in the list contains the 'USD' currency. Substring
 The 'USDMXN' symbol at index 68 in the list contains the 'USD' currency. Substring
 The 'EURMXN' symbol at index 69 in the list contains the 'EUR' currency. Substring
 The 'USDCOP' symbol at index 75 in the list contains the 'USD' currency. Substring
 The 'USDARS' symbol at index 76 in the list contains the 'USD' currency. Substring
 The 'USDCLP' symbol at index 77 in the list contains the 'USD' currency. Substring
 The 'EURSGD' symbol at index 89 in the list contains the 'EUR' currency. Substring
 The 'USDILS' symbol at index 95 in the list contains the 'USD' currency. Substring
 The 'USDTHB' symbol at index 122 in the list contains the 'USD' currency. Substrin
 The 'USDRMB' symbol at index 123 in the list contains the 'USD' currency. Substrin
 The 'EURILS' symbol at index 126 in the list contains the 'EUR' currency. Substrin
 The 'EURCNH' symbol at index 137 in the list contains the 'EUR' currency. Substrin
 The 'USDBRL' symbol at index 139 in the list contains the 'USD' currency. Substrin
 */
}
```

See also

StringSubstr, StringGetCharacter, StringLen, StringLen

StringGetCharacter

Returns value of a symbol, located in the specified position of a string.

```
ushort  StringGetCharacter(
   string  string_value,     // string
   int     pos               // symbol position in the string
   );
```

Parameters

string_value

[in] String.

pos

[in] Position of a symbol in the string. Can be from 0 to StringLen(text) -1.

Return Value

Symbol code or 0 in case of an error. To get the error code call GetLastError().

Example:

```
void OnStart()
  {
//--- delete all comments on the chart
   Comment("");
//--- declare a string, from which we will obtain the values of symbol codes and reme
   string message = "The script demonstrates the operation of the StringGetCharacter()
   int    length  = StringLen(message);
//--- declare a string variable, to which we will add the obtained symbols from the de
   string text    = "";
//--- in the loop by the demo string length
   for(int i=0; i<length; i++)
     {
      //--- wait 1/10 seconds
      Sleep(100);
      //--- get a symbol from a string located at the loop index in the demo string
      ushort char_code=StringGetCharacter(message, i);
      //--- add a symbol to the displayed string and display the resulting string as
      text+=ShortToString(char_code);
      Comment(text);
     }
//--- wait two seconds and remove the comment from the chart
   Sleep(2000);
   Comment("");

   /*
   Result: the demo string appears on the screen character by character
   The script demonstrates the operation of the StringGetCharacter() function
   */
  }
```

See also

StringSetCharacter, StringBufferLen, StringLen, StringFill, StringInit, StringToCharArray, StringToShortArray

StringInit

Initializes a string by specified symbols and provides the specified string size.

```
bool  StringInit(
   string&    string_var,        // string to initialize
   int        new_len=0,         // required string length after initialization
   ushort     character=0        // symbol, by which the string will be filled
   );
```

Parameters

string_var

[in][out] String that should be initialized and deinitialized.

new_len=0

[in] String length after initialization. If length=0, it deinitializes the string, i.e. the string buffer is cleared and the buffer address is zeroed.

character=0

[in] Symbol to fill the string.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Note

If character=0 and the length new_len>0, the buffer of the string of indicated length will be distributed and filled by zeroes. The string length will be equal to zero, because the whole buffer is filled out by string terminators.

Example:

```
void OnStart()
  {
//---
   string str;
   StringInit(str,200,0);
   Print("str = ",str,": StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/*  Result
str = : StringBufferLen(str) = 200   StringLen(str) = 0
*/
```

See also

StringBufferLen, StringLen

StringLen

Returns the number of symbols in a string.

```
int  StringLen(
   string  string_value      // string
   );
```

Parameters

string_value

[in] String to calculate length.

Return Value

Number of symbols in a string without the ending zero.

Example:

```
void OnStart()
  {
//--- define the test string
   string text="123456789012345";
//--- get the number of symbols in the string
   int str_len=StringLen(text);
//--- display the string and the number of symbols in it in the log
   PrintFormat("The StringLen() function returned the value of %d chars in string: '%s

   /*
   Result
   The StringLen() function returned the value of 15 chars in string: '123456789012345
   */
  }
```

See also

[StringBufferLen](), [StringTrimLeft](), [StringTrimRight](), [StringToCharArray](), [StringToShortArray]()

StringSetLength

Sets a specified length (in characters) for a string.

```
bool  StringSetLength(
   string&     string_var,       // string
   uint        new_length        // new string length
   );
```

Parameters

string_var

[in][out] String, for which a new length in characters should be set.

new_capacity

[in] Required string length in characters. If new_length is less than the current size, the excessive characters are discarded.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

TheStringSetLength() function does not change the size of the buffer allocated for a string.

Example:

```
void OnStart()
  {
//--- define the string
   string text="123456789012345";

//--- display a string and its length in the log
   PrintFormat("Before StringSetLength() the string '%s' has a size of %d characters",

//--- reduce the string size to 10 characters
   StringSetLength(text, 10);

//--- display a string, changed due to StringSetLength() operation, and its new lengtl
   PrintFormat("After StringSetLength() the string is now '%s', and has a size of %d c

   /*
   Result
   Before StringSetLength() the string '123456789012345' has a size of 15 characters
   After StringSetLength() the string is now '1234567890', and has a size of 10 charac
   */
  }
```

See also

StringLen, StringBufferLen, StringReserve StringInit, StringSetCharacter

StringReplace

It replaces all the found substrings of a string by a set sequence of symbols.

```
int  StringReplace(
   string&         str,              // the string in which substrings will be replace
   const string    find,             // the searched substring
   const string    replacement       // the substring that will be inserted to the fou
   );
```

Parameters

str

[in][out] The string in which you are going to replace substrings.

find

[in] The desired substring to replace.

replacement

[in] The string that will be inserted instead of the found one.

Return Value

The function returns the number of replacements in case of success, otherwise -1. To get an error code call the GetLastError() function.

Note

If the function has run successfully but no replacements have been made (the substring to replace was not found), it returns 0.

The error can result from incorrect *str* or *find* parameters (empty or non-initialized string, see StringInit()). Besides, the error occurs if there is not enough memory to complete the replacement.

Example:

```
   string text="The quick brown fox jumped over the lazy dog.";
   int replaced=StringReplace(text,"quick","slow");
   replaced+=StringReplace(text,"brown","black");
   replaced+=StringReplace(text,"fox","bear");
   Print("Replaced: ", replaced,". Result=",text);

//  Result
//  Replaced: 3. Result=The slow black bear jumped over the lazy dog.
//
```

See also

StringSetCharacter(), StringSubstr()

StringReserve

Reserves the buffer of a specified size for a string in memory.

```
bool  StringReserve(
   string&     string_var,        // string
   uint        new_capacity       // buffer size for storing a string
   );
```

Parameters

string_var

[in][out] String the buffer size should change the size for.

new_capacity

[in] Buffer size required for a string. If the new_capacity size is less than the string length, the size of the current buffer does not change.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

Generally, the string size is not equal to the size of the buffer meant for storing the string. When creating a string, the appropriate buffer is usually allocated with a margin. The StringReserve() function allows managing the buffer size and specify the optimal size for future operations.

Unlike StringInit(), the StringReserve() function does not change the string contents and does not fill it with characters.

Example:

```
void OnStart()
  {
   string s;
//--- check the operation speed without using StringReserve
   ulong t0=GetMicrosecondCount();
   for(int i=0; i< 1024; i++)
      s+=" "+(string)i;
   ulong msc_no_reserve=GetMicrosecondCount()-t0;
   s=NULL;
//--- now, let's do the same using StringReserve
   StringReserve(s,1024 * 3);
   t0=GetMicrosecondCount();
   for(int i=0; i< 1024; i++)
      s+=" "+(string)i;
   ulong msc_reserve=GetMicrosecondCount()-t0;
//--- check the time
   Print("Test with StringReserve passed for "+(string)msc_reserve+" msc");
   Print("Test without StringReserve passed for "+(string)msc_no_reserve+" msc");
/* Result
```

```
     Test with StringReserve passed for 50 msc
     Test without StringReserve passed for 121 msc
*/
  }
```

See also

StringBufferLen, StringSetLength, StringInit, StringSetCharacter

StringSetCharacter

Returns copy of a string with a changed character in a specified position.

```
bool  StringSetCharacter(
   string&    string_var,        // string
   int        pos,               // position
   ushort     character          // character
   );
```

Parameters

string_var

[in][out] String.

pos

[in] Position of a character in a string. Can be from 0 to StringLen(text).

character

[in] Symbol code Unicode.

Return Value

In case of success returns true, otherwise false. In order to get an error code, the GetLastError() function should be called.

Note

If pos is less than string length and the symbol code value = 0, the string is cut off (but the buffer size, distributed for the string remains unchanged). The string length becomes equal to pos.

If *pos* is equal to string length, the specified symbol is added at the string end, and the length is enlarged by one.

Example:

```
void OnStart()
  {
   string str="0123456789";
   Print("before: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- add zero value in the middle
   StringSetCharacter(str,6,0);
   Print("after: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- add symbol at the end
   int size=StringLen(str);
   StringSetCharacter(str,size,'+');
   Print("addition: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/* Result
   before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 10
   after:  str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
   addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
*/
```

See also

StringBufferLen, StringLen, StringFill, StringInit, CharToString, ShortToString, CharArrayToString, ShortArrayToString

StringSplit

Gets substrings by a specified separator from the specified string, returns the number of substrings obtained.

```
int  StringSplit(
   const string   string_value,       // A string to search in
   const ushort   separator,          // A separator using which substrings will be se
   string         & result[]          // An array passed by reference to get the founc
   );
```

Parameters

string_value

[in] The string from which you need to get substrings. The string will not change.

pos

[in] The code of the separator character. To get the code, you can use the StringGetCharacter() function.

result[]

[out] An array of strings where the obtained substrings are located.

Return Value

The number of substrings in the result[] array. If the separator is not found in the passed string, only one source string will be placed in the array.

If string_value is empty or NULL, the function will return zero. In case of an error the function returns -1. To get the error code, call the GetLastError() function.

Example:

```
string to_split="_life_is_good_"; // A string to split into substrings
   string sep="_";                // A separator as a character
   ushort u_sep;                  // The code of the separator character
   string result[];               // An array to get strings
   //--- Get the separator code
   u_sep=StringGetCharacter(sep,0);
   //--- Split the string to substrings
   int k=StringSplit(to_split,u_sep,result);
   //--- Show a comment
   PrintFormat("Strings obtained: %d. Used separator '%s' with the code %d",k,sep,u_se
   //--- Now output all obtained strings
   if(k>0)
     {
      for(int i=0;i<k;i++)
        {
         PrintFormat("result[%d]=\"%s\"",i,result[i]);
        }
     }
```

See also

StringReplace(), StringSubstr(), StringConcatenate()

StringSubstr

Extracts a substring from a text string starting from the specified position.

```
string  StringSubstr(
   string  string_value,      // string
   int     start_pos,         // position to start with
   int     length=-1          // length of extracted string
   );
```

Parameters

string_value

[in] String to extract a substring from.

start_pos

[in] Initial position of a substring. Can be from 0 to StringLen(text) -1.

length=-1

[in] Length of an extracted substring. If the parameter value is equal to -1 or parameter isn't set, the substring will be extracted from the indicated position till the string end.

Return Value

Copy of a extracted substring, if possible. Otherwise returns an empty string.

Example:

```
void OnStart()
  {
//--- get the name of the current symbol
   string name = Symbol();

//--- get the base and quoted symbol currencies
   string base   = StringSubstr(name, 0, 3);
   string quoted = StringSubstr(name, 3, 3);

//--- display the obtained symbol currencies in the log
   PrintFormat("Symbol: %s. Currency base: %s, currency quoted: %s", name, base, quote

   /*
   Result
   Symbol: EURUSD. Currency base: EUR, currency quoted: USD
   */
  }
```

See also

StringSplit, StringFind, StringGetCharacter

StringToLower

Transforms all symbols of a selected string into lowercase.

```
bool  StringToLower(
   string&  string_var      // string to process
   );
```

Parameters

string_var

[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Example:

```
void OnStart()
  {
//--- define the source string in uppercase
   string text=" - THIS STRING, WRITTEN IN UPPERCASE, MUST BE WRITTEN IN LOWERCASE";
//--- Display the source string in the log
   Print("Source line:\n", text);
//--- convert all string characters to lowercase and display the result in the log
   if(StringToLower(text))
      Print("The original string after using the StringToLower() function:\n", text);

   /*
   Result
   Source line:
    - THIS STRING, WRITTEN IN UPPERCASE, MUST BE WRITTEN IN LOWERCASE
   The original string after using the StringToLower() function:
    - this string, written in uppercase, must be written in lowercase
   */
  }
```

See also

StringToUpper, StringTrimLeft, StringTrimRight

StringToUpper

Transforms all symbols of a selected string into capitals.

```
bool  StringToUpper(
   string&  string_var      // string to process
   );
```

Parameters

string_var

[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Example:

```
void OnStart()
  {
//--- define the source string in lowercase
   string text=" - this string, written in lowercase, must be written in uppercase";
//--- Display the source string in the log
   Print("Source line:\n", text);
//--- convert all string characters to uppercase and display the result in the log
   if(StringToUpper(text))
      Print("The original string after using the StringToUpper() function:\n", text);

   /*
   Result
   Source line:
    - this string, written in lowercase, must be written in uppercase
   The original string after using the StringToUpper() function:
    - THIS STRING, WRITTEN IN LOWERCASE, MUST BE WRITTEN IN UPPERCASE
   */
  }
```

See also

StringToLower, StringTrimLeft, StringTrimRight

StringTrimLeft

The function cuts line feed characters, spaces and tabs in the left part of the string till the first meaningful symbol. The string is modified at place.

```
int  StringTrimLeft(
   string&  string_var      // string to cut
   );
```

Parameters

string_var

[in][out] String that will be cut from the left.

Return Value

Returns the number of cut symbols.

Example:

```
void OnStart()
  {
//--- define the source string with six spaces on the left
   string text="      All spaces on the left will be removed from this string";
//--- Display the source string in the log
   PrintFormat("Source line:\n'%s'", text);
//--- remove all spaces on the left and display the number of removed characters and t
   int num=StringTrimLeft(text);
   PrintFormat("The StringTrimLeft() function removed %d chars from the left side. No

   /*
   Result
   Source line:
   '      All spaces on the left will be removed from this string'
   The StringTrimLeft() function removed 6 chars from the left side. Now the line loo
   'All spaces on the left will be removed from this string'
   */
  }
```

See also

[StringTrimRight](), [StringToLower](), [StringToUpper]()

StringTrimRight

The function cuts line feed characters, spaces and tabs in the right part of the string after the last meaningful symbol. The string is modified at place.

```
int  StringTrimRight(
   string&  string_var      // string to cut
   );
```

Parameters

string_var

[in][out] String that will be cut from the right.

Return Value

Returns the number of cut symbols.

Example:

```
void OnStart()
  {
//--- define the source string with six spaces on the right
   string text="All spaces on the right will be removed from this string      ";
//--- Display the source string in the log
   PrintFormat("Source line:\n'%s'", text);
//--- remove all spaces on the right and display the number of removed characters and
   int num=StringTrimRight(text);
   PrintFormat("The StringTrimRight() function removed %d chars from the right side. N

   /*
   Result
   Source line:
   'All spaces on the right will be removed from this string      '
   The StringTrimRight() function removed 6 chars from the right side. Now the line lo
   'All spaces on the right will be removed from this string'
   */
  }
```

See also

[StringTrimLeft](), [StringToLower](), [StringToUpper]()

Date and Time

This is the group of functions for working with data of datetime type (an integer that represents the number of seconds elapsed from 0 hours of January 1, 1970).

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

Function	Action
TimeCurrent	Returns the last known server time (time of the last quote receipt) in the datetime format
TimeTradeServer	Returns the current calculation time of the trade server
TimeLocal	Returns the local computer time in datetime format
TimeGMT	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running
TimeDaylightSavings	Returns the sign of Daylight Saving Time switch
TimeGMTOffset	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch
TimeToStruct	Converts a datetime value into a variable of MqlDateTime structure type
StructToTime	Converts a variable of MqlDateTime structure type into a datetime value

TimeCurrent

Returns the last known server time, time of the last quote receipt for one of the symbols selected in the "Market Watch" window. In the OnTick() handler, this function returns the time of the received handled tick. In other cases (for example, call in handlers OnInit(), OnDeinit(), OnTimer() and so on) this is the time of the last quote receipt for any symbol available in the "Market Watch" window, the time shown in the title of this window. The time value is formed on a trade server and does not depend on the time settings on your computer. There are 2 variants of the function.

Call without parameters

```
datetime  TimeCurrent();
```

Call with MqlDateTime type parameter

```
datetime  TimeCurrent(
   MqlDateTime&  dt_struct      // structure type variable
   );
```

Parameters

dt_struct

[out] MqlDateTime structure type variable.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the strategy tester, TimeCurrent() is simulated according to historical data.

Example:

```
void OnStart()
  {
//--- declare the MqlDateTime variable to be filled with date/time data and get the ti
   MqlDateTime tm={};
   datetime    time1=TimeCurrent();    // first form of call: time of the last quote r
   datetime    time2=TimeCurrent(tm);  // second form of call: time of the last quote

//--- display the result of receiving the date/time and filling the structure with the
   PrintFormat("Tick time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u\n-
               (string)time1, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.day
   /*
   result:
   Tick time: 2024.04.18 15:40:06
   - Year: 2024
   - Month: 04
```

```
    - Day: 18
    - Hour: 15
    - Min: 40
    - Sec: 06
    - Day of Year: 108
    - Day of Week: 4 (THURSDAY)
    */
  }
```

TimeTradeServer

Returns the calculated current time of the trade server. Unlike TimeCurrent(), the calculation of the time value is performed in the client terminal and depends on the time settings on your computer. There are 2 variants of the function.

Call without parameters

```
datetime  TimeTradeServer();
```

Call with MqlDateTime type parameter

```
datetime  TimeTradeServer(
   MqlDateTime&  dt_struct      // Variable of structure type
   );
```

Parameters

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the strategy tester, TimeTradeServer() is simulated according to historical data and always equal to TimeCurrent().

Example:

```
void OnStart()
  {
//--- declare the MqlDateTime variable to be filled with date/time data and get the ti
   MqlDateTime tm={};
   datetime    time_current=TimeCurrent();                  // first form of call: ti
   datetime    time_server =TimeTradeServer(tm);            // second form of call: es
   int         difference  =int(time_current-time_server);  // difference between Time

//--- display the time of the last quote and the estimated current time of the trade s
   PrintFormat("Time Current: %s\nTime Trade Server: %s\n- Year: %u\n- Month: %02u\n-
               "- Hour: %02u\n- Min: %02u\n- Sec: %02u\n- Day of Year: %03u\n- Day of
               (string)time_current, (string)time_server, tm.year, tm.mon, tm.day, tm.
               EnumToString((ENUM_DAY_OF_WEEK)tm.day_of_week), difference);
   /*
   result:
   Time Current: 2024.04.18 16:10:14
   Time Trade Server: 2024.04.18 16:10:15
```

```
    - Year: 2024
    - Month: 04
    - Day: 18
    - Hour: 16
    - Min: 10
    - Sec: 15
    - Day of Year: 108
    - Day of Week: 4 (THURSDAY)
    Difference between Time Current and Time Trade Server: -1
    */
  }
```

TimeLocal

Returns the local time of a computer, where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime  TimeLocal();
```

Call with MqlDateTime type parameter

```
datetime  TimeLocal(
   MqlDateTime&  dt_struct      // Variable of structure type
   );
```

Parameters

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the strategy tester, TimeLocal() is always equal to TimeCurrent() simulated server time.

Example:

```
void OnStart()
  {
//--- declare the MqlDateTime variable to be filled with PC local time data
   MqlDateTime tm={};
   datetime    time1=TimeLocal();   // first form of call: PC local time
   datetime    time2=TimeLocal(tm); // second form of call: PC local time with fillin

//--- display the result of receiving PC local time and filling the structure with the
   PrintFormat("Local time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u\
               (string)time1, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.da
   /*
   result:
   Local time: 2024.04.18 19:44:09
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 19
   - Min: 44
   - Sec: 09
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   */
  }
```

TimeGMT

Returns the GMT, which is calculated taking into account the DST switch by the local time on the computer where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime  TimeGMT();
```

Call with MqlDateTime type parameter

```
datetime  TimeGMT(
   MqlDateTime&  dt_struct      // Variable of structure type
   );
```

Parameters

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the strategy tester, TimeGMT() is always equal to TimeTradeServer() simulated server time.

Example:

```
void OnStart()
  {
//--- declare the MqlDateTime variable to be filled with date/time data and get the P
   MqlDateTime tm={};
   datetime    time1=TimeLocal();             // first form of call: PC local time
   datetime    time2=TimeGMT(tm);             // second form of call: GMT calculated f
   int         shift=int(time1-time2)/3600;   // local time offset relative to GMT

//--- display local time and GMT with the data of the filled MqlDateTime structure in
   PrintFormat("Time Local: %s\nTime GMT: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\
               "- Hour: %02u\n- Min: %02u\n- Sec: %02u\n- Day of Year: %03u\n- Day of
               (string)time1, (string)time2, tm.year, tm.mon, tm.day, tm.hour, tm.min,
               EnumToString((ENUM_DAY_OF_WEEK)tm.day_of_week), shift);
   /*
   result:
   Time Local: 2024.04.18 19:37:23
   Time GMT: 2024.04.18 12:37:23
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 12
   - Min: 37
   - Sec: 23
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   Local time offset relative to GMT: +7
   */
  }
```

TimeDaylightSavings

Returns correction for daylight saving time in seconds, if the switch to summer time has been made. It depends on the time settings of your computer.

```
int  TimeDaylightSavings();
```

Return Value

If switch to winter (standard) time has been made, it returns 0.

Example:

```
void OnStart()
  {
//--- get daylight saving time adjustment in seconds
   int sec_dl=TimeDaylightSavings();

//--- create the text describing the received value
   string text=(sec_dl==0 ? "Standard \"winter\" time is used" :
                StringFormat("Daylight saving time has been switched over. The correct

//--- display a description of the daylight saving time adjustment in seconds in the
   Print(text);
   /*
   result for "winter" time:
   Standard "winter" time is used

   result for "summer" time:
   Daylight saving time has been switched over. The correction is -3600 seconds
   */
  }
```

TimeGMTOffset

Returns the current difference between GMT time and the local computer time in seconds, taking into account switch to winter or summer time. Depends on the time settings of your computer.

```
int  TimeGMTOffset();
```

Return Value

The value of int type, representing the current difference between GMT time and the local time of the computer TimeLocal in seconds.

```
TimeGMTOffset() =  TimeGMT() - TimeLocal()
```

Example:

```
void OnStart()
  {
//--- get local time, GMT and the difference between GMT and local computer time in se
   datetime time_local=TimeLocal();
   datetime time_gmt  =TimeGMT();
   int      offset    =TimeGMTOffset();

//--- show the obtained time and offset values in the log
   PrintFormat("Local Time: %s, GMT Time: %s, Seconds Offset: %+d", (string)time_local
   /*
   result:
   Local Time: 2024.04.18 19:35:52, GMT Time: 2024.04.18 12:35:52, Seconds Offset: -25
   */
  }
```

TimeToStruct

Converts a value of datetime type (number of seconds since 01.01.1970) into a structure variable MqlDateTime.

```
bool  TimeToStruct(
   datetime       dt,            // date and time
   MqlDateTime&   dt_struct      // structure for the adoption of values
   );
```

Parameters

dt

[in] Date value to convert.

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

True if successful, otherwise false. To get information about the error, call the GetLastError() function.

Example:

```
void OnStart()
  {
//--- get the last known time of the server, declare the date/time structure and fill
   datetime    time=TimeCurrent();
   MqlDateTime tm   ={};
   if(!TimeToStruct(time,tm))
      Print("TimeToStruct() failed. Error ", GetLastError());

//--- display the obtained server time and the result of filling the MqlDateTime struc
   PrintFormat("Server time: %s\n- Year: %u\n- Month: %02u\n- Day: %02u\n- Hour: %02u
               (string)time, tm.year, tm.mon, tm.day, tm.hour, tm.min, tm.sec, tm.day_
   /*
   result:
   Server time: 2024.04.18 15:47:27
   - Year: 2024
   - Month: 04
   - Day: 18
   - Hour: 15
   - Min: 47
   - Sec: 27
   - Day of Year: 108
   - Day of Week: 4 (THURSDAY)
   */
  }
```

StructToTime

Converts a structure variable MqlDateTime into a value of datetime type and returns the resulting value.

```
datetime  StructToTime(
   MqlDateTime&  dt_struct      // structure of the date and time
   );
```

Parameters

dt_struct

[in] Variable of structure type MqlDateTime.

Return Value

The value of datetime type containing the number of seconds since 01.01.1970.

Example:

```
#property script_show_inputs

input int   InpYear  =  0;     // Year
input int   InpMonth =  0;     // Month
input int   InpDay   =  0;     // Day
input int   InpHour  =  0;     // Hour
input int   InpMin   =  0;     // Minutes
input int   InpSec   =  0;     // Seconds

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- adjust the entered values and write them to variables
   int year =  (InpYear<1970 ? 1970 : InpYear); // If the year entered is less than 1
   int month=  (InpMonth<1 ?  1  :  InpMonth > 12  ?  12 :  InpMonth);
   int day  =  (InpDay   <1 ?  1  :  InpDay    > 31  ?  31 :  InpDay);
   int hour =  (InpHour  <0 ?  0  :  InpHour   > 23  ?  23 :  InpHour);
   int min  =  (InpMin   <0 ?  0  :  InpMin    > 59  ?  59 :  InpMin);
   int sec  =  (InpSec   <0 ?  0  :  InpSec    > 59  ?  59 :  InpSec);

//--- display the entered values in the log
   PrintFormat("Entered date and time: %04u.%02u.%02u %02u:%02u:%02u", InpYear, InpMo

//--- display the adjusted entered values in the log
   PrintFormat("Corrected date and time: %04u.%02u.%02u %02u:%02u:%02u", year, month,

//--- write the input values to the corresponding structure fields
   MqlDateTime time_struct={};
   time_struct.year= year;
```

```
    time_struct.mon = month;
    time_struct.day = day;
    time_struct.hour= hour;
    time_struct.min = min;
    time_struct.sec = sec;

 //--- convert the date and time from the structure into a variable of datetime type ar
    datetime time = StructToTime(time_struct);

 //--- display the result of conversion from an MqlDateTime structure type variable to
    Print("Converted date and time: ",TimeToString(time,TIME_DATE|TIME_MINUTES|TIME_SE
    /*
    results if zero default values are entered:
    Entered date and time: 0000.00.00 00:00:00
    Corrected date and time: 1970.01.01 00:00:00
    Converted date and time: 1970.01.01 00:00:00

    results if the wrong day of the current year's February is entered:
    Entered date and time: 2024.02.31 00:00:00
    Corrected date and time: 2024.02.31 00:00:00
    Converted date and time: 2024.03.02 00:00:00
    */
   }
```

Account Information

Functions that return parameters of the current account.

Function	Action
AccountInfoDouble	Returns a value of double type of the corresponding account property
AccountInfoInteger	Returns a value of integer type (bool, int or long) of the corresponding account property
AccountInfoString	Returns a value string type corresponding account property

AccountInfoDouble

Returns the value of the appropriate account property.

```
double  AccountInfoDouble(
   ENUM_ACCOUNT_INFO_DOUBLE  property_id      // Property identifier
   );
```

Parameters

property_id

[in] Property identifier. The value can be one of the values of ENUM_ACCOUNT_INFO_DOUBLE.

Return Value

Value of double type.

Example:

```
void OnStart()
  {
//--- Show all the information available from the function AccountInfoDouble()
   printf("ACCOUNT_BALANCE =  %G",AccountInfoDouble(ACCOUNT_BALANCE));
   printf("ACCOUNT_CREDIT =  %G",AccountInfoDouble(ACCOUNT_CREDIT));
   printf("ACCOUNT_PROFIT =  %G",AccountInfoDouble(ACCOUNT_PROFIT));
   printf("ACCOUNT_EQUITY =  %G",AccountInfoDouble(ACCOUNT_EQUITY));
   printf("ACCOUNT_MARGIN =  %G",AccountInfoDouble(ACCOUNT_MARGIN));
   printf("ACCOUNT_MARGIN_FREE =  %G",AccountInfoDouble(ACCOUNT_MARGIN_FREE));
   printf("ACCOUNT_MARGIN_LEVEL =  %G",AccountInfoDouble(ACCOUNT_MARGIN_LEVEL));
   printf("ACCOUNT_MARGIN_SO_CALL = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL));
   printf("ACCOUNT_MARGIN_SO_SO = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_SO));
  }
```

See also

SymbolInfoDouble, SymbolInfoString, SymbolInfoInteger, PrintFormat

AccountInfoInteger

Returns the value of the appropriate account property.

```
long  AccountInfoInteger(
   ENUM_ACCOUNT_INFO_INTEGER  property_id      // Identifier of the property
   );
```

Parameters

property_id

[in] Property identifier. The value can be one of the values of ENUM_ACCOUNT_INFO_INTEGER.

Return Value

Value of long type.

Note

The property must be of bool, int or long type.

Example:

```
void OnStart()
  {
//--- Show all the information available from the function AccountInfoInteger()
   printf("ACCOUNT_LOGIN =  %d",AccountInfoInteger(ACCOUNT_LOGIN));
   printf("ACCOUNT_LEVERAGE =  %d",AccountInfoInteger(ACCOUNT_LEVERAGE));
   bool thisAccountTradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_ALLOWED);
   bool EATradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_EXPERT);
   ENUM_ACCOUNT_TRADE_MODE tradeMode=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOU
   ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger

//--- Inform about the possibility to perform a trade operation
   if(thisAccountTradeAllowed)
      Print("Trade for this account is permitted");
   else
      Print("Trade for this account is prohibited!");

//--- Find out if it is possible to trade on this account by Expert Advisors
   if(EATradeAllowed)
      Print("Trade by Expert Advisors is permitted for this account");
   else
      Print("Trade by Expert Advisors is prohibited for this account!");

//--- Find out the account type
   switch(tradeMode)
     {
      case(ACCOUNT_TRADE_MODE_DEMO):
         Print("This is a demo account");
         break;
      case(ACCOUNT_TRADE_MODE_CONTEST):
          Print("This is a competition account");
          break;
       default:Print("This is a real account!");
      }

//--- Find out the StopOut level setting mode
   switch(stopOutMode)
      {
       case(ACCOUNT_STOPOUT_MODE_PERCENT):
          Print("The StopOut level is specified percentage");
          break;
       default:Print("The StopOut level is specified in monetary terms");
      }
   }
```

See also

Account Information

AccountInfoString

Returns the value of the appropriate account property.

```
string  AccountInfoString(
   ENUM_ACCOUNT_INFO_STRING  property_id      // Property identifier
   );
```

Parameters

property_id

[in] Property identifier. The value can be one of the values of ENUM_ACCOUNT_INFO_STRING.

Return Value

Value of string type.

Example:

```
void OnStart()
  {
//--- Show all the information available from the function AccountInfoString()
   Print("The name of the broker = ",AccountInfoString(ACCOUNT_COMPANY));
   Print("Deposit currency = ",AccountInfoString(ACCOUNT_CURRENCY));
   Print("Client name = ",AccountInfoString(ACCOUNT_NAME));
   Print("The name of the trade server = ",AccountInfoString(ACCOUNT_SERVER));
  }
```

See also

Account Information

State Checking

Functions that return parameters of the current state of the client terminal

Function	Action
GetLastError	Returns the last error
IsStopped	Returns true, if an mql5 program has been commanded to stop its operation
UninitializeReason	Returns the code of the reason for deinitialization
TerminalInfoInteger	Returns an integer value of a corresponding property of the mql5 program environment
TerminalInfoDouble	Returns a double value of a corresponding property of the mql5 program environment
TerminalInfoString	Returns a string value of a corresponding property of the mql5 program environment
MQLInfoInteger	Returns an integer value of a corresponding property of a running mql5 program
MQLInfoString	Returns a string value of a corresponding property of a running mql5 program
Symbol	Returns the name of a symbol of the current chart
Period	Returns the current chart timeframe
Digits	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol
Point	Returns the point size of the current symbol in the quote currency

GetLastError

Returns the contents of the system variable _LastError.

```
int  GetLastError();
```

Return Value

Returns the value of the last error that occurred during the execution of an mql5 program.

Note

After the function call, the contents of _LastError are not reset. To reset this variable, you need to call ResetLastError().

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlRates rates[1]={};   // display the current bar data here

//--- intentionally call a function with inappropriate parameters
   int res=CopyRates(NULL, PERIOD_CURRENT, 0, 2, rates);
   if(res!=2)
      PrintFormat("CopyRates() returned %d. LastError %d", res, GetLastError());

//--- reset the last error code before copying the current bar data to the MqlRates st
   ResetLastError();
//--- if the function does not work correctly, the error code will differ from 0
   CopyRates(NULL, PERIOD_CURRENT, 0, 1, rates);
   Print("CopyRates() error ", GetLastError());

//--- print the array of obtained values
   ArrayPrint(rates);
  }
```

See also

Trade Server Return Codes

IsStopped

Checks the forced shutdown of an mql5 program.

```
bool  IsStopped();
```

Return Value

Returns true, if the _StopFlag system variable contains a value other than 0. A nonzero value is written into _StopFlag, if a mql5 program has been commanded to complete its operation. In this case, you must immediately terminate the program, otherwise the program will be completed forcibly from the outside after 3 seconds.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- in an endless loop with a stop check
   while(!IsStopped())
     {
      //--- display the local PC time on the chart
      Comment("Time Local: ", TimeToString(TimeLocal(), TIME_DATE|TIME_MINUTES|TIME_SE
      Sleep(16);
     }
   Print("The StopFlag is set. The program will be stopped.");

//--- clean up
   Comment("");
  }
```

UninitializeReason

Returns the code of a reason for deinitialization.

```
int  UninitializeReason();
```

Return Value

Returns the value of _UninitReason which is formed before OnDeinit() is called. Value depends on the reasons that led to deinitialization.

Example:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- get the deinitialization reason code into the 'reason' variable
   int reason=UninitializeReason();
//--- create a message string with a deinitialization reason and send the message to t
   string message=StringFormat("%s: Uninitialize reason code: %d (%s)",__FUNCTION__, r
   Print(message);
//--- successful
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- create a message string with a deinitialization reason code from the 'reason' fo
   string message=StringFormat("%s: Uninitialize reason code: %d (%s)",__FUNCTION__, r
   Print(message);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
  }
//+------------------------------------------------------------------+
//| Return a description of the deinitialization reason code         |
//+------------------------------------------------------------------+
string UninitializeReasonDescription(const int reason)
  {
   switch(reason)
     {
      //--- the EA has stopped working calling the ExpertRemove() function
      case REASON_PROGRAM :
        return("Expert Advisor terminated its operation by calling the ExpertRemove()
      //--- program removed from a chart
      case REASON_REMOVE :
        return("Program has been deleted from the chart");
      //--- program recompiled
      case REASON_RECOMPILE :
        return("Program has been recompiled");
      //--- symbol or chart period changed
      case REASON_CHARTCHANGE :
        return("Symbol or chart period has been changed");
      //--- chart closed
      case REASON_CHARTCLOSE :
        return("Chart has been closed");
      //--- inputs changed by user
      case REASON_PARAMETERS :
        return("Input parameters have been changed by a user");
      //--- another account has been activated or reconnection to the trade server has
      case REASON_ACCOUNT :
        return("Another account has been activated or reconnection to the trade server
      //--- another chart template applied
      case REASON_TEMPLATE :
        return("A new template has been applied");
      //--- OnInit() handler returned a non-zero value
      case REASON_INITFAILED :
        return("This value means that OnInit() handler has returned a nonzero value");
      //--- terminal closed
      case REASON_CLOSE :
        return("Terminal has been closed");
     }

//--- deinitialization reason unknown
   return("Unknown reason");
  }
```

TerminalInfoInteger

Returns the value of a corresponding property of the mql5 program environment.

```
int  TerminalInfoInteger(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_INTEGER enumeration.

Return Value

Value of int type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the build number of the running terminal and its "64-bit terminal" property
   int  build = TerminalInfoInteger(TERMINAL_BUILD);
   bool x64   = TerminalInfoInteger(TERMINAL_X64);

//--- print the obtained terminal data in the journal
   PrintFormat("MetaTrader 5 %s build %d", (x64 ? "x64" : "x32"), build);
   /*
   result:
   MetaTrader 5 x64 build 4330
   */
  }
```

TerminalInfoDouble

Returns the value of a corresponding property of the mql5 program environment.

```
double  TerminalInfoDouble(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_DOUBLE enumeration.

Return Value

Value of double type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the build number of the running terminal and its "64-bit terminal" property
   int  build = TerminalInfoInteger(TERMINAL_BUILD);
   bool x64   = TerminalInfoInteger(TERMINAL_X64);

//--- print the obtained terminal data in the journal
   PrintFormat("MetaTrader 5 %s build %d", (x64 ? "x64" : "x32"), build);
   /*
   result:
   MetaTrader 5 x64 build 4330
   */
  }
```

TerminalInfoString

Returns the value of a corresponding property of the mql5 program environment. The property must be of string type.

```
string  TerminalInfoString(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_STRING enumeration.

Return Value

Value of string type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get OS and terminal data
   string os_ver  = TerminalInfoString(TERMINAL_OS_VERSION);      // user's OS
   string name    = TerminalInfoString(TERMINAL_NAME);            // terminal name
   string path    = TerminalInfoString(TERMINAL_PATH);            // folder the termi
   string data    = TerminalInfoString(TERMINAL_DATA_PATH);       // folder for stori
   string common  = TerminalInfoString(TERMINAL_COMMONDATA_PATH); // common folder fo

//--- send the obtained data to the journal
   PrintFormat("OS: %s\nTerminal: %s\n- Path: %s\n- Data path: %s\n- Common Data path
   /*
   OS: Windows 10 build 19045
   Terminal: MetaTrader 5
   - Path: E:\MetaQuotes\MetaTrader 5
   - Data path: E:\MetaQuotes\MetaTrader 5
   - Common Data path: C:\Users\admin\AppData\Roaming\MetaQuotes\Terminal\Common
   */
  }
```

MQLInfoInteger

Returns the value of a corresponding property of a running mql5 program.

```
int  MQLInfoInteger(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of values of the ENUM_MQL_INFO_INTEGER enumeration.

Return Value

Value of int type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the values of available and consumed memory for the MQL program
   int limit = MQLInfoInteger(MQL_MEMORY_LIMIT);   // maximum possible amount of dynar
   int used  = MQLInfoInteger(MQL_MEMORY_USED);    // memory size used by MQL5 prograr

//--- send the received values to the journal
   PrintFormat("Maximum possible amount of dynamic memory for MQL5 program: %d Mb\n"+
               "Memory used by MQL5 program: %d Mb", limit, used);
   /*
   result
   Maximum possible amount of dynamic memory for MQL5 program: 8388608 Mb
   Memory used by MQL5 program: 2 Mb
   */
  }
```

MQLInfoString

Returns the value of a corresponding property of a running mql5 program.

```
string  MQLInfoString(
   int  property_id      // Identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the ENUM_MQL_INFO_STRING enumeration.

Return Value

Value of string type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the launched program data
   string name = MQLInfoString(MQL_PROGRAM_NAME);  // name of the launched MQL5 progra
   string path = MQLInfoString(MQL_PROGRAM_PATH);  // running program path

//--- send the obtained data to the journal
   PrintFormat("Name of the running MQL program: '%s'\nPath of the running MQL progra
   /*
   result:
   Name of the running MQL program: 'MQLInfoString'
   Path of the running MQL program: E:\MetaQuotes\MetaTrader 5\MQL5\Scripts\MQLInfoStr
   */
  }
```

Symbol

Returns the name of a symbol of the current chart.

```
string  Symbol();
```

Return Value

Value of the _Symbol system variable, which stores the name of the current chart symbol.

Note

Unlike Expert Advisors, indicators and scripts, services are not bound to a specific chart. Therefore, Symbol() returns an empty string ("") for a service.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart symbol name
   string name = Symbol();

//--- send the obtained data to the journal
   PrintFormat("Current chart symbol name: '%s'", name);
   /*
   result
   Current chart symbol name: 'EURUSD'
   */
  }
```

Period

Returns the current chart timeframe.

```
ENUM_TIMEFRAMES  Period();
```

Return Value

The contents of the _Period variable that contains the value of the current chart timeframe. The value can be one of the values of the ENUM_TIMEFRAMES enumeration.

Note

Unlike Expert Advisors, indicators and scripts, services are not bound to a specific chart. Therefore, Period() returns 0 for a service.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the timeframe value of the current chart and its description
   ENUM_TIMEFRAMES period    = Period();
   string          timeframe = StringSubstr(EnumToString(period), 7);

//--- send the obtained data to the journal
   PrintFormat("Current chart timeframe: %s\nTimeframe value: %s (%d)",
               timeframe, EnumToString(period), period);
   /*
   result:
   Current chart timeframe: H4
   Timeframe value: PERIOD_H4 (16388)
   */
  }
```

See also

PeriodSeconds, Chart timeframes, Date and Time, Visibility of objects

Digits

Returns the number of decimal digits determining the accuracy of price of the current chart symbol.

```
int  Digits();
```

Return Value

The value of the _Digits variable which stores the number of decimal digits determining the accuracy of price of the current chart symbol.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of decimal places for the current chart symbol
   int digits = Digits();

//--- send the obtained data to the journal
   Print("Number of decimal digits for the current chart symbol: ", digits);
   /*
   result:
   Number of decimal digits for the current chart symbol: 5
   */
  }
```

Point

Returns the point size of the current symbol in the quote currency.

```
double  Point();
```

Return Value

The value of the _Point variable which stores the point size of the current symbol in the quote currency.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current symbol point size in the quote currency
   double point = Point();

//--- send the obtained data to the journal
   Print("Point size of the current symbol in the quote currency: ", DoubleToString(po
   /*
   result:
   Point size of the current symbol in the quote currency: 0.00001
   */
  }
```

Event Handling

The MQL5 language provides handling of certain predefined events. The functions for handling these events should be defined in an MQL5 program: function name, return type, a set of parameters (if any) and their types should strictly correspond to the description of an event handling function.

The client terminal event handler uses the return and parameter types to identify functions processing an event. If a certain function has some parameters or a return type not corresponding to the descriptions below, such a function cannot be used for handling an event.

Function	Description
OnStart	The function is called when the Start event occurs to perform actions set in the script
OnInit	The function is called in indicators and EAs when the Init event occurs to initialize a launched MQL5 program
OnDeinit	The function is called in indicators and EAs when the Deinit event occurs to de-initialize a launched MQL5 program
OnTick	The function is called in EAs when the NewTick event occurs to handle a new quote
OnCalculate	The function is called in indicators when the Calculate event occurs to handle price data changes
OnTimer	The function is called in indicators and EAs during the Timer periodic event generated by the terminal at fixed time intervals
OnTrade	The function is called in EAs during the Trade event generated at the end of a trading operation on a trade server
OnTradeTransaction	The function is called in EAs when the TradeTransaction event occurs to process a trade request execution results
OnBookEvent	The function is called in EAs when the BookEvent event occurs to process changes in the market depth
OnChartEvent	The function is called in indicators and EAs when the ChartEvent event occurs to process chart changes made by a user or an MQL5 program
OnTester	The function is called in EAs when the Tester event occurs to perform necessary actions after testing an EA on history data
OnTesterInit	The function is called in EAs when the TesterInit event occurs to perform necessary actions before optimization in the strategy tester
OnTesterDeinit	The function is called in EAs when the TesterDeinit event occurs after EA optimization in the strategy tester
OnTesterPass	The function is called in EAs when the TesterPass even occurs to handle an arrival of a new data frame during EA optimization in the strategy tester

The client terminal sends incoming events to corresponding open charts. Also, events may be generated by charts (chart events) or mql5 programs (custom events). Generating graphical object creation/deletion events can be enabled/disabled by setting the CHART_EVENT_OBJECT_CREATE and CHART_EVENT_OBJECT_DELETE chart properties. Each mql5 application and chart have their own queue of events where all newly arrived events are placed.

A program gets events only from the chart it is running on. All events are handled one after another in the order of their receipt. If the queue already contains the NewTick event or this event is in the processing stage, then the new NewTick event is not added to mql5 application queue. Similarly, if the ChartEvent is already in an mql5 program queue or such an event is being handled, then a new event of this type is not placed into a queue. Timer event handling is processed in the same way – if the Timer event is already in the queue or is being handled, no new timer event is set into a queue.

Event queues have a limited but sufficient size, so the queue overflow is unlikely for a correctly developed program. When the queue overflows, new events are discarded without being set into a queue.

It is strongly recommended not to use infinite loops to handle events. Possible exceptions are scripts handling a single Start event.

Libraries do not handle any events.

OnStart

The function is called in scripts and services when the Start event occurs. The function is intended for one-time execution of actions implemented in a program. There are two function types.

The version that returns the result

```
int  OnStart(void);
```

Return Value

The value of int type displayed in the Journal tab.

The entry "script script_name removed (result code N)" is created in the terminal journal after a script execution is complete. Here N is a value returned by the OnStart() function.

The entry "service service_name stopped (result code N)" is created in the terminal journal after a service execution is complete. Here N is a value returned by the OnStart() function.

The OnStart() call that returns the execution result is recommended for use since it not only allows for a script or service execution, but also returns an error code or other useful data to analyze the program execution result.

The version without a result return is left only for compatibility with old codes. It is not recommended for use

```
void  OnStart(void);
```

Note

OnStart() is the only function for handling events in scripts and services. No other events are sent to these programs. In turn, the Start event is not passed to EAs and custom indicators.

Sample script:

```
//--- macros for working with colors
#define XRGB(r,g,b)    (0xFF000000|(uchar(r)<<16)|(uchar(g)<<8)|uchar(b))
#define GETRGB(clr)    ((clr)&0xFFFFFF)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set a downward candle color
   Comment("Set a downward candle color");
   ChartSetInteger(0,CHART_COLOR_CANDLE_BEAR,GetRandomColor());
   ChartRedraw(); // update the chart immediately without waiting for a new tick
   Sleep(1000);   // pause for 1 second to see all the changes
//--- set an upward candle color
   Comment("Set an upward candle color");
   ChartSetInteger(0,CHART_COLOR_CANDLE_BULL,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set the background color
```

```
   Comment("Set the background color");
   ChartSetInteger(0,CHART_COLOR_BACKGROUND,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of Ask line
   Comment("Set color of Ask line");
   ChartSetInteger(0,CHART_COLOR_ASK,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of Bid line
   Comment("Set color of Bid line");
   ChartSetInteger(0,CHART_COLOR_BID,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of a downward bar and a downward candle frame
   Comment("Set color of a downward bar and a downward candle frame");
   ChartSetInteger(0,CHART_COLOR_CHART_DOWN,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of a chart line and Doji candlesticks
   Comment("Set color of a chart line and Doji candlesticks");
   ChartSetInteger(0,CHART_COLOR_CHART_LINE,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of an upward bar and an upward candle frame
   Comment("Set color of an upward bar and an upward candle frame");
   ChartSetInteger(0,CHART_COLOR_CHART_UP,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of axes, scale and OHLC line
   Comment("Set color of axes, scale and OHLC line");
   ChartSetInteger(0,CHART_COLOR_FOREGROUND,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set a grid color
   Comment("Set a grid color");
   ChartSetInteger(0,CHART_COLOR_GRID,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set Last price color
   Comment("Set Last price color");
   ChartSetInteger(0,CHART_COLOR_LAST,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
//--- set color of Stop Loss and Take Profit order levels
   Comment("Set color of Stop Loss and Take Profit order levels");
   ChartSetInteger(0,CHART_COLOR_STOP_LEVEL,GetRandomColor());
   ChartRedraw();
   Sleep(1000);
```

```
//--- set color of volumes and market entry levels
   Comment("Set color of volumes and market entry levels");
   ChartSetInteger(0,CHART_COLOR_VOLUME,GetRandomColor());
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Return a randomly generated color                                |
//+------------------------------------------------------------------+
color GetRandomColor()
  {
   color clr=(color)GETRGB(XRGB(rand()%255,rand()%255,rand()%255));
   return clr;
  }
```

See also

Event handling functions, Program running, Client terminal events

OnInit

The function is called in indicators and EAs when the Init event occurs. It is used to initialize a running MQL5 program. There are two function types.

The **version that returns the result**

```
int  OnInit(void);
```

Return Value

int type value, zero means successful initialization.

When returning INIT_FAILED, the EA is forcibly unloaded from the chart.

When returning INIT_FAILED, the indicator is not unloaded from the chart. The indicator remaining on the chart is non-operational — event handlers are not called in the indicator.

The OnInit() call that returns the execution result is recommended for use since it not only allows for program initialization, but also returns an error code in case of an early program termination.

The version without a result return is left only for compatibility with old codes. It is not recommended for use

```
void  OnInit(void);
```

Note

The Init event is generated immediately after loading an EA or an indicator. The event is not generated for scripts. The OnInit() function is used to initialize an MQL5 program. If OnInit() has a return value of int type, the non-zero return code means failed initialization and generates the Deinit event with the REASON_INITFAILED deinitialization reason code.

OnInit() function of void type always means successful initialization and is not recommended for use.

For optimizing the EA inputs, it is recommended to use values from the ENUM_INIT_RETCODE enumeration as a return code. These values are intended for establishing the optimization process management, including selection of the most suitable test agents. It is possible to request data on agent configuration and resources (number of cores, free memory amount, etc.) using the TerminalInfoInteger() function during the EA initialization before launching the test. Based on the obtained data, you can either allow using the test agent or ban it from optimizing the EA.

ID	Description
INIT_SUCCEEDED	Initialization successful, EA test can be continued. This code means the same as the zero value – the EA initialization in the tester is successful.
INIT_FAILED	Initialization failed. There is no point in continuing the test due to unavoidable errors. For example, it is impossible to create an indicator necessary for the EA operation. The return of this value means the same as returning the value different from zero – EA initialization in the tester failed.

ID	Description
INIT_PARAMETERS_INCORRECT	Designed to denote an incorrect set of input parameters by a programmer. In the general optimization table, the result string with this return code is highlighted in red. A test for such a set of EA inputs is not performed. The agent is ready to receive a new task. When this value is received, the strategy tester does not pass this task to other agents for repeated execution.
INIT_AGENT_NOT_SUITABLE	No program execution errors during initialization. However, for some reasons, the agent is not suitable for conducting a test. For example, there is not enough RAM, no OpenCL support, etc. After returning this code, the agent no longer receives tasks until the very end of this optimization.

Using OnInit() returning INIT_FAILED/INIT_PARAMETERS_INCORRECT in the tester have some peculiarities that should be considered when optimizing EAs:

- the set of parameters the OnInit() returned INIT_PARAMETERS_INCORRECT for is considered unsuitable for testing and is not used to obtain the next population during genetic optimization. Too many "discarded" parameter sets may lead to incorrect results when searching for optimal EA parameters. The search algorithm assumes that the optimization criterion function is smooth and has no gaps on the entire multitude of input parameters.
- if OnInit() returns INIT_FAILED, this means that a test cannot be launched, and the EA is unloaded from the agent's memory. The EA is loaded again to perform the next pass with a new set of parameters. Launching the next optimization pass takes much more time as compared to calling TesterStop().

Sample OnInit() function for an EA

```
//--- input parameters
input int      ma_period=20; // moving average period

//--- handle of the indicator used in the EA
int indicator_handle;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check ma_period validity
   if(ma_period<=0)
     {
      PrintFormat("Invalid ma_period input value: %d",ma_period);
      return (INIT_PARAMETERS_INCORRECT);
     }
//--- during optimization
   if(MQLInfoInteger(MQL_OPTIMIZATION))
     {
      //--- check available RAM for the agent
      int available_memory_mb=TerminalInfoInteger(TERMINAL_MEMORY_TOTAL);
      if(available_memory_mb<2000)
        {
         PrintFormat("Insufficient memory for the test agent: %d MB",
                     available_memory_mb);
         return (INIT_AGENT_NOT_SUITABLE);
        }
     }
//--- check for the indicator
   indicator_handle=iCustom(_Symbol,_Period,"My_Indicator",ma_period);
   if(indicator_handle==INVALID_HANDLE)
     {
      PrintFormat("Failed to generate My_Indicator handle. Error code %d",
                  GetLastError());
      return (INIT_FAILED);
     }
//--- EA initialization successful
   return(INIT_SUCCEEDED);
  }
```

See also

OnDeinit, Event handling functions, Program running, Client terminal events, Initialization of variables, Creating and deleting objects

OnDeinit

The function is called in indicators and EAs when the Deinit event occurs. It is used to deinitialize a running MQL5 program.

```
void  OnDeinit(
   const int   reason         // deinitialization reason code
   );
```

Parameters

reason

[in] Deinitialization reason code.

Return Value

No return value

Note

Deinit event is generated for EAs and indicators in the following cases:

- before a re-initialization due to the change of a symbol or a chart period the mql5 program is attached to;
- before a re-initialization due to the change of the inputs;
- before unloading an mql5 program.

The *reason* parameter may have the following values:

Constant	Value	Description
REASON_PROGRAM	0	The EA has stopped working calling the ExpertRemove() function
REASON_REMOVE	1	Program removed from a chart
REASON_RECOMPILE	2	Program recompiled
REASON_CHARTCHANGE	3	A symbol or a chart period is changed
REASON_CHARTCLOSE	4	Chart closed
REASON_PARAMETERS	5	Inputs changed by a user
REASON_ACCOUNT	6	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings
REASON_TEMPLATE	7	Another chart template applied
REASON_INITFAILED	8	The OnInit() handler returned a non-zero value
REASON_CLOSE	9	Terminal closed

EA deinitialization reason codes can be received by the UninitializeReason() function or from the predefined _UninitReason variable.

Sample OnInit() and OnDeinit() functions for the EA

```
input int fake_parameter=3;      // useless parameter
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Get the number of a build where the program is compiled
   Print(__FUNCTION__," Build #",__MQLBUILD__);
//--- Reset reason code can also be obtained in OnInit()
   Print(__FUNCTION__," Deinitialization reason code can be received during the EA res
//--- The first way to get a deinitialization reason code
   Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
//--- The second way to get a deinitialization reason code
   Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReas
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- The first way to get a deinitialization reason code
   Print(__FUNCTION__," Deinitialization reason code = ",reason);
//--- The second way to get a deinitialization reason code
   Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
//--- The third way to get a deinitialization reason code
   Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReas
  }
//+------------------------------------------------------------------+
//| Return a textual description of the deinitialization reason code |
//+------------------------------------------------------------------+
string getUninitReasonText(int reasonCode)
  {
   string text="";
//---
   switch(reasonCode)
     {
      case REASON_ACCOUNT:
         text="Account was changed";break;
      case REASON_CHARTCHANGE:
         text="Symbol or timeframe was changed";break;
      case REASON_CHARTCLOSE:
         text="Chart was closed";break;
      case REASON_PARAMETERS:
         text="Input-parameter was changed";break;
      case REASON_RECOMPILE:
         text="Program "+__FILE__+" was recompiled";break;
      case REASON_REMOVE:
         text="Program "+__FILE__+" was removed from chart";break;
      case REASON_TEMPLATE:
         text="New template was applied to chart";break;
      default:text="Another reason";
     }
//---
   return text;
  }
```

See also

OnInit, Event handling functions, Program running, Client terminal events, Uninitialization reason codes, Visibility scope and lifetime of variables, Creating and deleting objects

OnTick

The function is called in EAs when the NewTick event occurs to handle a new quote.

```
void  OnTick(void);
```

Return Value

No return value

Note

The NewTick event is generated only for EAs upon receiving a new tick for a symbol of the chart the EA is attached to. There is no point in defining the OnTick() function in a custom indicator or a script since a NewTick event is not generated for them.

The Tick event is generated only for EAs, but this does not mean that EAs have to feature the OnTick() function, since Timer, BookEvent and ChartEvent events are also generated for EAs in addition to NewTick.

All events are handled one after another in the order of their receipt. If the queue already contains the NewTick event or this event is in the processing stage, then the new NewTick event is not added to mql5 application queue.

The NewTick event is generated regardless of whether auto trading is enabled (AutoTrading button). Disabled auto trading means only a ban on sending trade requests from an EA. The EA operation is not stopped.

Disabling auto trading by pressing the AutoTrading button does not interrupt the current execution of the OnTick() function.

Example of the EA featuring its entire trading logic in the OnTick() function

```
//+------------------------------------------------------------------+
//|                                                   TradeByATR.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA trading in the \"explosive\" candle direction"
#property description "\"Explosive\" candle has the body size exceeding k*ATR"
#property description "The \"revers\" parameter reverses the signal direction"

input double lots=0.1;          // volume in lots
input double kATR=3;            // signal candle length in ATR
input int    ATRperiod=20;      // ATR indicator period
input int    holdbars=8;        // number of bars to hold position on
input int    slippage=10;       // allowable slippage
input bool   revers=false;      // reverse the signal?
input ulong  EXPERT_MAGIC=0;    // EA's MagicNumber
//--- for storing the ATR indicator handle
```

```
int atr_handle;
//--- here we will store the last ATR values and the candle body
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- initialize global variables
   last_atr=0;
   last_body=0;
//--- set the correct volume
   double min_lot=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   trade_lot=lots>min_lot? lots:min_lot;
//--- create ATR indicator handle
   atr_handle=iATR(_Symbol,_Period,ATRperiod);
   if(atr_handle==INVALID_HANDLE)
     {
      PrintFormat("%s: failed to create iATR, error code %d",__FUNCTION__,GetLastErro
      return(INIT_FAILED);
     }
//--- successful EA initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- inform of the EA operation end code
   Print(__FILE__,": Deinitialization reason code = ",reason);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- trading signal
   static int signal=0; // +1 means a buy signal, -1 means a sell signal
//--- check and close old positions opened more than 'holdbars' bars ago
   ClosePositionsByBars(holdbars,slippage,EXPERT_MAGIC);
//--- check for a new bar
   if(isNewBar())
     {
      //--- check for a signal presence
      signal=CheckSignal();
     }
```

```
//--- if a netting position is opened, skip the signal - wait till it closes
   if(signal!=0 && PositionsTotal()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
     {
      signal=0;
      return; // exit the NewTick event handler and do not enter the market before a r
     }
//--- for a hedging account, each position is held and closed separately
   if(signal!=0)
     {
      //--- buy signal
      if(signal>0)
        {
         PrintFormat("%s: Buy signal! Revers=%s",__FUNCTION__,string(revers));
         if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
      //--- sell signal
      if(signal<0)
        {
         PrintFormat("%s: Sell signal! Revers=%s",__FUNCTION__,string(revers));
         if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
     }
//--- OnTick function end
  }
//+------------------------------------------------------------------+
//| Check for a new trading signal                                   |
//+------------------------------------------------------------------+
int CheckSignal()
  {
//--- 0 means no signal
   int res=0;
//--- get ATR value on a penultimate complete bar (the bar index is 2)
   double atr_value[1];
   if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
     {
      last_atr=atr_value[0];
      //--- get data on the last closed bar to the MqlRates type array
      MqlRates bar[1];
      if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
        {
         //--- calculate the bar body size on the last complete bar
         last_body=bar[0].close-bar[0].open;
         //--- if the body of the last bar (with index 1) exceeds the previous ATR val
         if(MathAbs(last_body)>kATR*last_atr)
            res=last_body>0?1:-1; // positive value for the upward candle
        }
      else
```

```
         PrintFormat("%s: Failed to receive the last bar! Error",__FUNCTION__,GetLast
     }
   else
      PrintFormat("%s: Failed to receive ATR indicator value! Error",__FUNCTION__,Get
//--- if reverse trading mode is enabled
   res=revers?-res:res;  // reverse the signal if necessary (return -1 instead of 1 ar
//--- return a trading signal value
   return (res);
  }
//+------------------------------------------------------------------+
//|  Return 'true' when a new bar appears                            |
//+------------------------------------------------------------------+
bool isNewBar(const bool print_log=true)
  {
   static datetime bartime=0; // store open time of the current bar
//--- get open time of the zero bar
   datetime currbar_time=iTime(_Symbol,_Period,0);
//--- if open time changes, a new bar has arrived
   if(bartime!=currbar_time)
     {
      bartime=currbar_time;
      lastbar_timeopen=bartime;
      //--- display data on open time of a new bar in the log
      if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION)||MQLInfoInteger(MQL_TESTER)))
        {
         //--- display a message with a new bar open time
         PrintFormat("%s: new bar on %s %s opened at %s",__FUNCTION__,_Symbol,
                     StringSubstr(EnumToString(_Period),7),
                     TimeToString(TimeCurrent(),TIME_SECONDS));
         //--- get data on the last tick
         MqlTick last_tick;
         if(!SymbolInfoTick(Symbol(),last_tick))
            Print("SymbolInfoTick() failed, error = ",GetLastError());
         //--- display the last tick time up to milliseconds
         PrintFormat("Last tick was at %s.%03d",
                     TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%100
        }
      //--- we have a new bar
      return (true);
     }
//--- no new bar
   return (false);
  }
//+------------------------------------------------------------------+
//| Buy at a market price with a specified volume                    |
//+------------------------------------------------------------------+
bool Buy(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- buy at a market price
```

```
   return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| Sell at a market price with a specified volume                   |
//+------------------------------------------------------------------+
bool Sell(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- sell at a market price
   return (MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| Close positions by hold time in bars                             |
//+------------------------------------------------------------------+
void ClosePositionsByBars(int holdtimebars,ulong deviation=10,ulong  magicnumber=0)
  {
   int total=PositionsTotal(); // number of open positions
//--- iterate over open positions
   for(int i=total-1; i>=0; i--)
     {
      //--- position parameters
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
      int bars=iBarShift(_Symbol,PERIOD_CURRENT,position_open)+1;

      //--- if a position's lifetime is already large, while MagicNumber and a symbol
      if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
         int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
         double volume=PositionGetDouble(POSITION_VOLUME);
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         string str_type=StringSubstr(EnumToString(type),14);
         StringToLower(str_type); // lower the text case for correct message formatti
         PrintFormat("Close position #%I64u %s %s %.2f",
                     position_ticket,position_symbol,str_type,volume);
         //--- set an order type and sending a trade request
         if(type==POSITION_TYPE_BUY)
            MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber,position_ticket);
         else
            MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber,position_ticket);
        }
     }
  }
//+------------------------------------------------------------------+
//| Prepare and send a trade request                                 |
//+------------------------------------------------------------------+
bool MarketOrder(ENUM_ORDER_TYPE type,double volume,ulong slip,ulong magicnumber,ulong
  {
```

```
//--- declaring and initializing structures
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(type==ORDER_TYPE_BUY)
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- request parameters
   request.action   =TRADE_ACTION_DEAL;                     // trading operation type
   request.position =pos_ticket;                            // position ticket if clos
   request.symbol   =Symbol();                              // symbol
   request.volume   =volume;                                // volume
   request.type     =type;                                  // order type
   request.price    =price;                                 // trade price
   request.deviation=slip;                                  // allowable deviation fro
   request.magic    =magicnumber;                           // order MagicNumber
//--- send a request
   if(!OrderSend(request,result))
     {
      //--- display data on failure
      PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
                  request.symbol,EnumToString(type),volume,request.price,GetLastError
      return (false);
     }
//--- inform of a successful operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
   return (true);
  }
```

See also

Event handling functions, Program running, Client terminal events, OnTimer, OnBookEvent, OnChartEvent

OnCalculate

The function is called in the indicators when the Calculate event occurs for processing price data changes. There are two function types. Only one of them can be used within a single indicator.

Calculation based on data array

```
int  OnCalculate(
   const int        rates_total,       // price[] array size
   const int        prev_calculated,   // number of handled bars at the previous call
   const int        begin,             // index number in the price[] array meaningful
   const double&    price[]            // array of values for calculation
   );
```

Calculations based on the current timeframe timeseries

```
int  OnCalculate(
   const int        rates_total,       // size of input time series
   const int        prev_calculated,   // number of handled bars at the previous call
   const datetime&  time[],            // Time array
   const double&    open[],            // Open array
   const double&    high[],            // High array
   const double&    low[],             // Low array
   const double&    close[],           // Close array
   const long&      tick_volume[],     // Tick Volume array
   const long&      volume[],          // Real Volume array
   const int&       spread[]           // Spread array
   );
```

Parameters

rates_total

[in] Size of the price[] array or input series available to the indicator for calculation. In the second function type, the parameter value corresponds to the number of bars on the chart it is launched at.

prev_calculated

[in] Contains the value returned by the OnCalculate() function during the previous call. It is designed to skip the bars that have not changed since the previous launch of this function.

begin

[in] Index value in the price[] array meaningful data starts from. It allows you to skip missing or initial data, for which there are no correct values.

price[]

[in] Array of values for calculations. One of the price timeseries or a calculated indicator buffer can be passed as the price[] array. Type of data passed for calculation can be defined using the _AppliedTo predefined variable.

time{}

[in] Array with bar open time values.

open[]

[in] Array with Open price values.

high[]

[in] Array with High price values.

low[]

[in] Array with Low price values.

close[]

[in] Array with Close price values.

tick_volume[]

[in] Array with tick volume values.

volume[]

[in] Array with trade volume values.

spread[]

[in] Array with spread values for bars.

Return Value

int type value to be passed as the *prev_calculated* parameter during the next function call.

Note

If the OnCalculate() function is equal to zero, no indicator values are shown in the DataWindow of the client terminal.

If the price data have been changed since the last call of the OnCalculate() function (a deeper history has been loaded or gaps in the history have been filled), the value of the *prev_calculated* input parameter is set to zero by the terminal itself.

To define the indexing direction in the *time[]*, *open[]*, *high[]*, *low[]*, *close[]*, *tick_volume[]*, *volume[]* and *spread[]* arrays, call the ArrayGetAsSeries() function. In order not to depend on defaults, call the ArraySetAsSeries() function for the arrays to work with.

When using the first function type, a necessary timeseries or indicator is selected by a user as the price[] array in the Parameters tab when launching the indicator. To do this, specify the necessary element in the drop-down list of the "Apply to" field.

To get custom indicator values from other mql5 programs, the iCustom() function is used. It returns the indicator handle for subsequent operations. It is also possible to specify the required *price []* array or the handle of another indicator. This parameter should be passed the last in the list of input variables of a custom indicator.

It is necessary to use the connection between the value returned by the OnCalculate() function and the *prev_calculated* second input parameter. When calling the function, the *prev_calculated* parameter contains the value returned by the OnCalculate() function during the previous call. This makes it possible to implement resource-saving algorithms for calculating a custom indicator in order to avoid repetitive calculations for the bars that have not changed since the previous launch of this function.

Sample indicator

```
//+------------------------------------------------------------------+
//|                                           OnCalculate_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample Momentum indicator calculation"

//---- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_type1   DRAW_LINE
#property indicator_color1  Blue
//---- inputs
input int MomentumPeriod=14; // Calculation period
//---- indicator buffer
double    MomentumBuffer[];
//--- global variable for storing calculation period
int       IntPeriod;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- check the input parameter
   if(MomentumPeriod<0)
     {
      IntPeriod=14;
      Print("Period parameter has an incorrect value. The following value is to be use
     }
   else
      IntPeriod=MomentumPeriod;
//---- buffers
   SetIndexBuffer(0,MomentumBuffer,INDICATOR_DATA);
//---- indicator name to be displayed in DataWindow and subwindow
   IndicatorSetString(INDICATOR_SHORTNAME,"Momentum"+"("+string(IntPeriod)+")");
//--- set index of the bar the drawing starts from
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,IntPeriod-1);
//--- set 0.0 as an empty value that is not drawn
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//--- indicator accuracy to be displayed
   IndicatorSetInteger(INDICATOR_DIGITS,2);
  }
//+------------------------------------------------------------------+
//|  Momentum indicator calculation                                  |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,     // price[] array size
                const int prev_calculated, // number of previously handled bars
                const int begin,           // where significant data start from
                const double &price[])     // value array for handling
  {
//--- initial position for calculations
   int StartCalcPosition=(IntPeriod-1)+begin;
//---- if calculation data is insufficient
   if(rates_total<StartCalcPosition)
      return(0);  // exit with a zero value - the indicator is not calculated
//--- correct draw begin
   if(begin>0)
      PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,StartCalcPosition+(IntPeriod-1));
//--- start calculations, define the starting position
   int pos=prev_calculated-1;
   if(pos<StartCalcPosition)
      pos=begin+IntPeriod;
//--- main calculation loop
   for(int i=pos;i<rates_total && !IsStopped();i++)
      MomentumBuffer[i]=price[i]*100/price[i-IntPeriod];
//--- OnCalculate execution is complete. Return the new prev_calculated value for the
   return(rates_total);
  }
```

See also

ArrayGetAsSeries, ArraySetAsSeries, iCustom, Event handling functions, Program running, Client terminal events, Access to timeseries and indicators

OnTimer

The function is called in EAs during the Timer event generated by the terminal at fixed time intervals.

```
void  OnTimer(void);
```

Return Value

No return value

Note

The Timer event is periodically generated by the client terminal for an EA, which activated the timer using the EventSetTimer() function. Usually, this function is called in the OnInit() function. When the EA stops working, the timer should be eliminated using EventKillTimer(), which is usually called in the OnDeinit() function.

Each Expert Advisor and each indicator work with its own timer receiving events solely from this timer. During mql5 application shutdown, the timer is forcibly destroyed in case it has been created but has not been disabled by EventKillTimer() function.

If you need to receive timer events more frequently than once per second, use EventSetMillisecondTimer() for creating a high-resolution timer.

In general, when the timer period is reduced, the testing time is increased, as the handler of timer events is called more often. When working in real-time mode, timer events are generated no more than 1 time in 10-16 milliseconds due to hardware limitations.

Only one timer can be launched for each program. Each mql5 application and chart have their own queue of events where all newly arrived events are placed. If the queue already contains Timer event or this event is in the processing stage, then the new Timer event is not added to mql5 application queue.

Sample EA with the OnTimer() handler

```
//+------------------------------------------------------------------+
//|                                             OnTimer_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Example of using the timer for calculating the trading server t
#property description "It is recommended to run the EA at the end of a trading week be
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create a timer with a 1 second period
   EventSetTimer(1);
```

```
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy the timer after completing the work
   EventKillTimer();

  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- time of the OnTimer() first call
   static datetime start_time=TimeCurrent();
//--- trade server time during the first OnTimer() call
   static datetime start_tradeserver_time=0;
//--- calculated trade server time
   static datetime calculated_server_time=0;
//--- local PC time
   datetime local_time=TimeLocal();
//--- current estimated trade server time
   datetime trade_server_time=TimeTradeServer();
//--- if a server time is unknown for some reason, exit ahead of time
   if(trade_server_time==0)
      return;
//--- if the initial trade server value is not set yet
   if(start_tradeserver_time==0)
     {
      start_tradeserver_time=trade_server_time;
      //--- set a calculated value of a trade server
      Print(trade_server_time);
      calculated_server_time=trade_server_time;
     }
   else
     {
      //--- increase time of the OnTimer() first call
      if(start_tradeserver_time!=0)
         calculated_server_time=calculated_server_time+1;;
     }
//---
   string com=StringFormat("                  Start time: %s\r\n",TimeToString(start_t
   com=com+StringFormat("                  Local time: %s\r\n",TimeToString(local_time
   com=com+StringFormat("TimeTradeServer time: %s\r\n",TimeToString(trade_server_time,
   com=com+StringFormat(" EstimatedServer time: %s\r\n",TimeToString(calculated_server
//--- display values of all counters on the chart
   Comment(com);
  }
```

See also

EventSetTimer, EventSetMillisecondTimer, EventKillTimer, GetTickCount, GetMicrosecondCount, Client terminal events

OnTrade

The function is called in EAs when the Trade event occurs. The function is meant for processing changes in order, position and trade lists.

```
void  OnTrade(void);
```

Return Value

No return value

Note

OnTrade() is called only for Expert Advisors. It is not used in indicators and scripts even if you add there a function with the same name and type.

For any trade action (placing a pending order, opening/closing a position, placing stops, activating pending orders, etc.), the history of orders and trades and/or the list of positions and current orders is changed appropriately.

When handling an order, a trade server sends the terminal a message about the incoming Trade event. To retrieve relevant data on orders and trades from history, it is necessary to perform a trading history request using HistorySelect() first.

The trade events are generated by the server in case of:

- changing active orders,
- changing positions,
- changing deals,
- changing trade history.

Each Trade event may appear as a result of one or several trade requests. Trade requests are sent to the server using OrderSend() or OrderSendAsync(). Each request can lead to several trade events. You cannot rely on the statement "One request - one Trade event", since the processing of events may be performed in several stages, and each operation may change the state of orders, positions and the trade history.

OnTrade() handler is called after the appropriate OnTradeTransaction() calls. In general, there is no exact correlation in the number of OnTrade () and OnTradeTransaction () calls. One OnTrade() call corresponds to one or several OnTradeTransaction calls.

Sample EA with OnTrade() handler

```
//+------------------------------------------------------------------+
//|                                              OnTrade_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

input    int days=7;            // depth of trade history in days
```

```
//--- set the limits of the trade history on the global scope
datetime      start;              // start date for trade history in cache
datetime      end;                // end date for trade history in cache
//--- global counters
int           orders;             // number of active orders
int           positions;          // number of open positions
int           deals;              // number of deals in the trade history cache
int           history_orders;     // number of orders in the trade history cache
bool          started=false;      // flag of counter relevance

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   end=TimeCurrent();
   start=end-days*PeriodSeconds(PERIOD_D1);
   PrintFormat("Limits of the history to be loaded: start - %s, end - %s",
               TimeToString(start),TimeToString(end));
   InitCounters();
//---
   return(0);
  }
//+------------------------------------------------------------------+
//|  initialization of position, order and trade counters            |
//+------------------------------------------------------------------+
void InitCounters()
  {
   ResetLastError();
//--- load history
   bool selected=HistorySelect(start,end);
   if(!selected)
     {
      PrintFormat("%s. Failed to load history from %s to %s to cache. Error code: %d",
                  __FUNCTION__,TimeToString(start),TimeToString(end),GetLastError());
      return;
     }
//--- get the current value
   orders=OrdersTotal();
   positions=PositionsTotal();
   deals=HistoryDealsTotal();
   history_orders=HistoryOrdersTotal();
   started=true;
   Print("Counters of orders, positions and deals successfully initialized");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   if(started) SimpleTradeProcessor();
   else InitCounters();
  }
//+------------------------------------------------------------------+
//| called when a Trade event arrives                                |
//+------------------------------------------------------------------+
void OnTrade()
  {
   if(started) SimpleTradeProcessor();
   else InitCounters();
  }
//+------------------------------------------------------------------+
//| example of processing changes in trade and history               |
//+------------------------------------------------------------------+
void SimpleTradeProcessor()
  {
   end=TimeCurrent();
   ResetLastError();
//--- download trading history from the specified interval to the program cache
   bool selected=HistorySelect(start,end);
   if(!selected)
     {
      PrintFormat("%s. Failed to load history from %s to %s to cache. Error code: %d",
                  __FUNCTION__,TimeToString(start),TimeToString(end),GetLastError());
      return;
     }
//--- get the current values
   int curr_orders=OrdersTotal();
   int curr_positions=PositionsTotal();
   int curr_deals=HistoryDealsTotal();
   int curr_history_orders=HistoryOrdersTotal();
//--- check if the number of active orders has been changed
   if(curr_orders!=orders)
     {
      //--- number of active orders has been changed
      PrintFormat("Number of orders has been changed. Previous value is %d, current va
                  orders,curr_orders);
      //--- update the value
      orders=curr_orders;
     }
//--- changes in the number of open positions
   if(curr_positions!=positions)
     {
      //--- number of open positions has been changed
      PrintFormat("Number of positions has been changed. Previous value is %d, current
                  positions,curr_positions);
      //--- update the value
      positions=curr_positions;
     }
//--- changes in the number of deals in the trade history cache
   if(curr_deals!=deals)
     {
      //--- number of deals in the trade history cache has been changed
      PrintFormat("Number of deals has been changed. Previous value is %d, current val
                  deals,curr_deals);
      //--- update the value
      deals=curr_deals;
     }
//--- changes in the number of history orders in the trade history cache
   if(curr_history_orders!=history_orders)
     {
      //--- number of history orders in the trade history cache has been changed
      PrintFormat("Number of orders in history has been changed. Previous value is %d,
                  history_orders,curr_history_orders);
     //--- update the value
     history_orders=curr_history_orders;
     }
//--- checking if it is necessary to change the limits of the trade history to be requ
   CheckStartDateInTradeHistory();
  }
//+------------------------------------------------------------------+
//|  changing the start date for requesting the trade history        |
//+------------------------------------------------------------------+
void CheckStartDateInTradeHistory()
  {
//--- initial interval, if we were to start working right now
   datetime curr_start=TimeCurrent()-days*PeriodSeconds(PERIOD_D1);
//--- make sure that the start limit of the trade history has not gone
//--- more than 1 day over the intended date
   if(curr_start-start>PeriodSeconds(PERIOD_D1))
     {
      //--- correct the start date of history to be loaded in the cache
      start=curr_start;
      PrintFormat("New start limit of the trade history to be loaded: start => %s",
                  TimeToString(start));
      //--- now reload the trade history for the updated interval
      HistorySelect(start,end);
      //--- correct the deal and order counters in history for further comparison
      history_orders=HistoryOrdersTotal();
      deals=HistoryDealsTotal();
     }
  }
//+------------------------------------------------------------------+
/* Sample output:
  Limits of the history to be loaded: start - 2018.07.16 18:11, end - 2018.07.23 18:1
  The counters of orders, positions and deals are successfully initialized
  Number of orders has been changed. Previous value 0, current value 1
  Number of orders has been changed. Previous value 1, current value 0
  Number of positions has been changed. Previous value 0, current value 1
  Number of deals has been changed. Previous value 0, current value 1
  Number of orders in the history has been changed. Previous value 0, current value 1
*/
```

See also

OrderSend, OrderSendAsync, OnTradeTransaction, Client terminal events

OnTradeTransaction

The function is called in EAs when the TradeTransaction event occurs. The function is meant for handling trade request execution results.

```
void  OnTradeTransaction()
   const MqlTradeTransaction&    trans,     // trade transaction structure
   const MqlTradeRequest&        request,   // request structure
   const MqlTradeResult&         result     // response structure
   );
```

Parameters

trans

[in] MqlTradeTransaction type variable describing a transaction made on a trading account.

request

[in] MqlTradeRequest type variable describing a trade request that led to a transaction. It contains the values for TRADE_TRANSACTION_REQUEST type transaction only.

result

[in] MqlTradeResult type variable containing an execution result of a trade request that led to a transaction. It contains the values for TRADE_TRANSACTION_REQUEST type transaction only.

Return Value

No return value

Note

OnTradeTransaction() is called to handle the TradeTransaction event sent by the trade server to the terminal in the following cases:

- sending a trade request from an MQL5 program using the OrderSend()/OrderSendAsync() functions and its subsequent execution;
- sending a trade request manually via the GUI and its subsequent execution;
- activations of pending and stop orders on the server;
- performing operations on the trade server side.

Data on transaction type is contained in the *type* field of the *trans* variable. Types of trade transactions are described in the ENUM_TRADE_TRANSACTION_TYPE enumeration:

- TRADE_TRANSACTION_ORDER_ADD – adding a new active order
- TRADE_TRANSACTION_ORDER_UPDATE – changing an existing order
- TRADE_TRANSACTION_ORDER_DELETE – deleting an order from the list of active ones
- TRADE_TRANSACTION_DEAL_ADD – adding a deal to history
- TRADE_TRANSACTION_DEAL_UPDATE – changing a deal in history
- TRADE_TRANSACTION_DEAL_DELETE – deleting a deal from history
- TRADE_TRANSACTION_HISTORY_ADD – adding an order to history as a result of execution or cancelation
- TRADE_TRANSACTION_HISTORY_UPDATE – changing an order in the order history
- TRADE_TRANSACTION_HISTORY_DELETE – deleting an order from the order history
- TRADE_TRANSACTION_POSITION – position change not related to a trade execution

- TRADE_TRANSACTION_REQUEST – notification that a trade request has been processed by the server and the result of its processing has been received.

When handling transactions of TRADE_TRANSACTION_REQUEST type, it is necessary to analyze the second and third parameters of the OnTradeTransaction() function – *request* and *result* – to receive additional information.

Sending a buy trade request leads to a chain of trade transactions on a trading account: 1) request is accepted for processing, 2) an appropriate purchase order is created for the account, 3) the order is then executed, 4) the executed order is removed from the list of active ones, 5) adding to the history of orders, 6) the subsequent transaction is added to history and 7) a new position is created. All these stages are trade transactions. The arrival of each such transaction to the terminal is the TradeTransaction event. Priority of these transactions' arrival at the terminal is not guaranteed. Thus, you should not expect that one group of transactions will arrive after another one when developing your trading algorithm.

When transactions are processed by the EA's OnTradeTransaction() handler, the terminal goes on handling the incoming trade transactions. Thus, the trading account status may change at the course of OnTradeTransaction() operation. For example, while an MQL5 program handles adding a new order, it can be executed, deleted from the list of open orders and moved to history. The program is notified of all these events.

Transactions queue length comprises 1024 elements. If OnTradeTransaction() handles yet another transaction for too long, the previous ones can be superseded by new transactions in the queue.

OnTrade() handler is called after the appropriate OnTradeTransaction() calls. In general, there is no exact correlation in the number of OnTrade () and OnTradeTransaction () calls. One OnTrade() call corresponds to one or several OnTradeTransaction calls.

Each Trade event may appear as a result of one or several trade requests. Trade requests are sent to the server using OrderSend() or OrderSendAsync(). Each request can lead to several trade events. You cannot rely on the statement "One request - one Trade event", since the processing of events may be performed in several stages and each operation may change the state of orders, positions and the trade history.

Sample EA with OnTradeTransaction() handler

```
//+------------------------------------------------------------------+
//|                                     OnTradeTransaction_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample listener of TradeTransaction events"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   PrintFormat("LAST PING=%.f ms",
               TerminalInfoInteger(TERMINAL_PING_LAST)/1000.);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//---
   static int counter=0;   // counter of OnTradeTransaction() calls
   static uint lasttime=0; // time of the OnTradeTransaction() last call
//---
   uint time=GetTickCount();
//--- if the last transaction was performed more than 1 second ago,
   if(time-lasttime>1000)
     {
      counter=0; // then this is a new trade operation, an the counter can be reset
      if(IS_DEBUG_MODE)
         Print(" New trade operation");
     }
   lasttime=time;
   counter++;
   Print(counter,". ",__FUNCTION__);
//--- result of trade request execution
   ulong            lastOrderID   =trans.order;
   ENUM_ORDER_TYPE  lastOrderType =trans.order_type;
   ENUM_ORDER_STATE lastOrderState=trans.order_state;
//--- the name of the symbol, for which a transaction was performed
   string trans_symbol=trans.symbol;
//--- type of transaction
   ENUM_TRADE_TRANSACTION_TYPE  trans_type=trans.type;
   switch(trans.type)
     {
      case  TRADE_TRANSACTION_POSITION:   // position modification
        {
         ulong pos_ID=trans.position;
         PrintFormat("MqlTradeTransaction: Position  #%I64u %s modified: SL=%.5f TP=%
                     pos_ID,trans_symbol,trans.price_sl,trans.price_tp);
        }
      break;
      case TRADE_TRANSACTION_REQUEST:     // sending a trade request
         PrintFormat("MqlTradeTransaction: TRADE_TRANSACTION_REQUEST");
         break;
      case TRADE_TRANSACTION_DEAL_ADD:    // adding a trade
        {
         ulong          lastDealID   =trans.deal;
         ENUM_DEAL_TYPE lastDealType =trans.deal_type;
         double        lastDealVolume=trans.volume;
         //--- Trade ID in an external system - a ticket assigned by an exchange
         string Exchange_ticket="";
         if(HistoryDealSelect(lastDealID))
            Exchange_ticket=HistoryDealGetString(lastDealID,DEAL_EXTERNAL_ID);
         if(Exchange_ticket!="")
            Exchange_ticket=StringFormat("(Exchange deal=%s)",Exchange_ticket);

         PrintFormat("MqlTradeTransaction: %s deal #%I64u %s %s %.2f lot   %s",EnumToS
                     lastDealID,EnumToString(lastDealType),trans_symbol,lastDealVolume
        }
      break;
      case TRADE_TRANSACTION_HISTORY_ADD: // adding an order to the history
        {
         //--- order ID in an external system - a ticket assigned by an Exchange
         string Exchange_ticket="";
         if(lastOrderState==ORDER_STATE_FILLED)
           {
            if(HistoryOrderSelect(lastOrderID))
               Exchange_ticket=HistoryOrderGetString(lastOrderID,ORDER_EXTERNAL_ID);
            if(Exchange_ticket!="")
               Exchange_ticket=StringFormat("(Exchange ticket=%s)",Exchange_ticket);
           }
         PrintFormat("MqlTradeTransaction: %s order #%I64u %s %s %s   %s",EnumToString
                     lastOrderID,EnumToString(lastOrderType),trans_symbol,EnumToString
        }
      break;
      default: // other transactions
        {
         //--- order ID in an external system - a ticket assigned by Exchange
         string Exchange_ticket="";
         if(lastOrderState==ORDER_STATE_PLACED)
           {
            if(OrderSelect(lastOrderID))
               Exchange_ticket=OrderGetString(ORDER_EXTERNAL_ID);
            if(Exchange_ticket!="")
               Exchange_ticket=StringFormat("Exchange ticket=%s",Exchange_ticket);
           }
         PrintFormat("MqlTradeTransaction: %s order #%I64u %s %s   %s",EnumToString(tr
                     lastOrderID,EnumToString(lastOrderType),EnumToString(lastOrderSta
        }
      break;
     }
//--- order ticket
   ulong orderID_result=result.order;
   string retcode_result=GetRetcodeID(result.retcode);
   if(orderID_result!=0)
      PrintFormat("MqlTradeResult: order #%d retcode=%s ",orderID_result,retcode_resul
//---
  }
//+------------------------------------------------------------------+
//| convert numeric response codes to string mnemonics               |
//+------------------------------------------------------------------+
string GetRetcodeID(int retcode)
  {
   switch(retcode)
     {
      case 10004: return("TRADE_RETCODE_REQUOTE");             break;
      case 10006: return("TRADE_RETCODE_REJECT");              break;
      case 10007: return("TRADE_RETCODE_CANCEL");              break;
      case 10008: return("TRADE_RETCODE_PLACED");              break;
      case 10009: return("TRADE_RETCODE_DONE");                break;
      case 10010: return("TRADE_RETCODE_DONE_PARTIAL");        break;
      case 10011: return("TRADE_RETCODE_ERROR");               break;
      case 10012: return("TRADE_RETCODE_TIMEOUT");             break;
      case 10013: return("TRADE_RETCODE_INVALID");             break;
      case 10014: return("TRADE_RETCODE_INVALID_VOLUME");      break;
      case 10015: return("TRADE_RETCODE_INVALID_PRICE");       break;
      case 10016: return("TRADE_RETCODE_INVALID_STOPS");       break;
      case 10017: return("TRADE_RETCODE_TRADE_DISABLED");      break;
      case 10018: return("TRADE_RETCODE_MARKET_CLOSED");       break;
      case 10019: return("TRADE_RETCODE_NO_MONEY");            break;
      case 10020: return("TRADE_RETCODE_PRICE_CHANGED");       break;
      case 10021: return("TRADE_RETCODE_PRICE_OFF");           break;
      case 10022: return("TRADE_RETCODE_INVALID_EXPIRATION");  break;
      case 10023: return("TRADE_RETCODE_ORDER_CHANGED");       break;
      case 10024: return("TRADE_RETCODE_TOO_MANY_REQUESTS");   break;
      case 10025: return("TRADE_RETCODE_NO_CHANGES");          break;
      case 10026: return("TRADE_RETCODE_SERVER_DISABLES_AT");  break;
      case 10027: return("TRADE_RETCODE_CLIENT_DISABLES_AT");  break;
      case 10028: return("TRADE_RETCODE_LOCKED");              break;
      case 10029: return("TRADE_RETCODE_FROZEN");              break;
      case 10030: return("TRADE_RETCODE_INVALID_FILL");        break;
      case 10031: return("TRADE_RETCODE_CONNECTION");          break;
      case 10032: return("TRADE_RETCODE_ONLY_REAL");           break;
      case 10033: return("TRADE_RETCODE_LIMIT_ORDERS");        break;
      case 10034: return("TRADE_RETCODE_LIMIT_VOLUME");        break;
      case 10035: return("TRADE_RETCODE_INVALID_ORDER");       break;
      case 10036: return("TRADE_RETCODE_POSITION_CLOSED");     break;
```

```
         default:
            return("TRADE_RETCODE_UNKNOWN="+IntegerToString(retcode));
            break;
        }
//---
   }
```

See also

OrderSend, OrderSendAsync, OnTradeTransaction, Trade request structure, Trade transaction structure, Trade transaction types, Trade operation types, Client terminal events

OnBookEvent

The function is called in indicators and EAs when the BookEvent event occurs. It is meant for handling Depth of Market changes.

```
void  OnBookEvent(
   const string&   symbol         // symbol
   );
```

Parameters

symbol

[in] Name of a symbol the BookEvent has arrived for

Return Value

No return value

Note

To get the BookEvent events for any symbol, simply subscribe to receive them for this symbol using the MarketBookAdd() function. To cancel subscription for receiving the BookEvent for a certain symbol, call the MarketBookRelease() function.

The BookEvent broadcasts within the entire chart. This means that if one application on a chart subscribes to the BookEvent using the MarketBookAdd function, all other indicators and EAs launched on the same chart and having the OnBookEvent() handler receive this event as well. Therefore, it is necessary to analyze a symbol name passed to the OnBookEvent() handler as the *symbol* parameter.

Separate BookEvent counters sorted by symbols are provided for all applications running on the same chart. This means that each chart may have multiple subscriptions to different symbols, and a counter is provided for each symbol. Subscribing and unsubscribing from BookEvent changes the subscription counter for specified symbols only within one chart. In other words, there may be two adjacent charts to the BookEvent for the same symbol but different subscription counter values.

The initial subscription counter value is zero. At each MarketBookAdd() call, the subscription counter for a specified symbol on the chart is increased by one (chart symbol and symbol in MarketBookAdd() do not have to match). When calling MarketBookRelease(), the counter of subscriptions for a specified symbol within the chart is decreased by one. The BookEvent events for any symbol are broadcast within the chart till the counter is equal to zero. Therefore, it is important that each MQL5 program that contains MarketBookAdd () calls correctly unsubscribes from getting events for each symbol using MarketBookRelease () at the end of its work. To achieve this, the number of MarketBookAdd() and MarketBookRelease() calls should be even for each call during the entire MQL5 program lifetime. Using flags or custom subscription counters within the program allows you to safely work with BookEvent events and prevents disabling subscriptions for getting this event in third-party programs within the same chart.

BookEvent events are never skipped and are always placed into a queue even if handling the previous BookEvent handling is not over yet.

Example

```
//+------------------------------------------------------------------+
//|                                          OnBookEvent_Sample.mq5 |
//|                       Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com/en/articles/2635"
#property version   "1.00"
#property description "Example of measuring the market depth refresh rate using OnBook
#property description "The code is taken from the article https://www.mql5.com/en/arti
//--- input parameters
input ulong ExtCollectTime   =30;  // test time in seconds
input ulong ExtSkipFirstTicks=10;  // number of ticks skipped at start
//--- flag of subscription to BookEvent events
bool book_subscribed=false;
//--- array for accepting requests from the market depth
MqlBookInfo  book[];
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- show the start
   Comment(StringFormat("Waiting for the first %I64u ticks to arrive",ExtSkipFirstTick
   PrintFormat("Waiting for the first %I64u ticks to arrive",ExtSkipFirstTicks);
//--- enable market depth broadcast
   if(MarketBookAdd(_Symbol))
     {
      book_subscribed=true;
      PrintFormat("%s: MarketBookAdd(%s) function returned true",__FUNCTION__,_Symbol)
     }
   else
      PrintFormat("%s: MarketBookAdd(%s) function returned false! GetLastError()=%d",_
//--- successful initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Deinitialize expert                                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- display deinitialization reason code
   Print(__FUNCTION__,": Deinitialization reason code = ",reason);
//--- cancel subscription for getting market depth events
   if(book_subscribed)
     {
      if(!MarketBookRelease(_Symbol))
         PrintFormat("%s: MarketBookRelease(%s) returned false! GetLastError()=%d",_Sy
      else
```

```
         book_subscribed=false;
     }
//---
  }
//+------------------------------------------------------------------+
//| BookEvent function                                               |
//+------------------------------------------------------------------+
void OnBookEvent(const string &symbol)
  {
   static ulong starttime=0;             // test start time
   static ulong tickcounter=0;           // market depth update counter
//--- work with depth market events only if we subscribed to them ourselves
   if(!book_subscribed)
      return;
//--- count updates only for a certain symbol
   if(symbol!=_Symbol)
      return;
//--- skip first ticks to clear the queue and to prepare
   tickcounter++;
   if(tickcounter<ExtSkipFirstTicks)
      return;
//--- remember the start time
   if(tickcounter==ExtSkipFirstTicks)
      starttime=GetMicrosecondCount();
//--- request for the market depth data
   MarketBookGet(symbol,book);
//--- when to stop?
   ulong endtime=GetMicrosecondCount()-starttime;
   ulong ticks  =1+tickcounter-ExtSkipFirstTicks;
// how much time has passed in microseconds since the start of the test?
   if(endtime>ExtCollectTime*1000*1000)
     {
      PrintFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endtime/1000.
      ExpertRemove();
      return;
     }
//--- display the counters in the comment field
   if(endtime>0)
      Comment(StringFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endt
  }
```

See also

MarketBookAdd, MarketBookRelease, MarketBookGet, OnTrade, OnTradeTransaction, OnTick, Event handling functions, Program running, Client terminal events

OnChartEvent

The function is called in indicators and EAs when the ChartEvent event occurs. The function is meant for handling chart changes made by a user or an MQL5 program.

```
void  OnChartEvent()
   const int        id,        // event ID
   const long&      lparam,    // long type event parameter
   const double&    dparam,    // double type event parameter
   const string&    sparam     // string type event parameter
   );
```

Parameters

id

[in] Event ID from the ENUM_CHART_EVENT enumeration.

lparam

[in] long type event parameter

dparam

[in] double type event parameter

sparam

[in] string type event parameter

Return Value

No return value

Note

There are 11 types of events that can be handled using the predefined OnChartEvent() function. 65535 IDs from CHARTEVENT_CUSTOM to CHARTEVENT_CUSTOM_LAST inclusive are provided for custom events. To generate a custom event, use the EventChartCustom() function.

Short event description from the ENUM_CHART_EVENT enumeration:

- CHARTEVENT_KEYDOWN — pressing a key on the keyboard when a chart window is in focus;
- CHARTEVENT_MOUSE_MOVE — moving the mouse and mouse button clicks (if CHART_EVENT_MOUSE_MOVE=true for a chart);
- CHARTEVENT_OBJECT_CREATE — create a graphical object (if CHART_EVENT_OBJECT_CREATE=true for a chart);
- CHARTEVENT_OBJECT_CHANGE — change object properties via the properties dialog;
- CHARTEVENT_OBJECT_DELETE — delete a graphical object (if CHART_EVENT_OBJECT_DELETE=true for a chart);
- CHARTEVENT_CLICK — clicking on a chart;
- CHARTEVENT_OBJECT_CLICK — mouse click on a graphical object belonging to a chart;
- CHARTEVENT_OBJECT_DRAG — dragging a graphical object with a mouse;
- CHARTEVENT_OBJECT_ENDEDIT — finish editing text in the Edit input box of a graphical object (OBJ_EDIT);
- CHARTEVENT_CHART_CHANGE — change a chart;

- CHARTEVENT_CUSTOM+n — custom event ID, where n is within the range from 0 to 65535. CHARTEVENT_CUSTOM_LAST contains the last acceptable custom event ID (CHARTEVENT_CUSTOM+65535).

All MQL5 programs work in threads other than the main thread of the application. The main application thread is responsible for handling all Windows system messages and, in its turn, generates Windows messages for its own application as a result of this handling. For example, moving the mouse on a chart (WM_MOUSE_MOVE event) generates several system messages for subsequent rendering of the application window, and also sends internal messages to experts and indicators launched on the chart. A situation may occur, where the main application thread has not yet processed the WM_PAINT system message (and therefore has not yet rendered the modified chart), while an EA or an indicator has already received the mouse movement event. In this case, the chart property CHART_FIRST_VISIBLE_BAR will be changed only after the chart is rendered.

For each event type, the inputs of the OnChartEvent() function have certain values necessary for handling that event. The table lists events and values passed via the parameters.

Event	'id' parameter value	'lparam' parameter value	'dparam' parameter value	'sparam' parameter value
Keystroke event	CHARTEVENT_KEYDOWN	pressed button code	The number of keypresses generated while the key was held in the pressed state	String value of the bit mask, which describes the status of the keyboard keys
Mouse events (if CHART_EVENT_MOUSE_MOVE=true for a chart)	CHARTEVENT_MOUSE_MOVE	X coordinate	Y coordinate	String value of the bit mask, which describes the status of the mouse keys
Mouse wheel event (if CHART_EVENT_MOUSE_WHEEL=true for the chart)	CHARTEVENT_MOUSE_WHEEL	Flags of states of keys and mouse buttons, X and Y coordinates of the cursor. See the description in the example	The Delta value of the mouse wheel scroll	—
Creating a graphical object (if CHART_EVENT_OBJECT_CREATE=true for a chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of a created graphical object
Changing object properties via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of a changed graphical object

Event	'id' parameter value	'lparam' parameter value	'dparam' parameter value	'sparam' parameter value
Removing a graphical object (if CHART_EVENT_OBJECT_DELETE=true for a chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of a removed graphical object
Mouse click on a chart	CHARTEVENT_CLICK	X coordinate	Y coordinate	—
Mouse click on a graphical object	CHARTEVENT_OBJECT_CLICK	X coordinate	Y coordinate	Name of a graphical object the event has occurred on
Moving a graphical object with mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of a moved graphical object
Finishing a text editing in the "Input field" graphical object input box	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the "Input field" graphical object, in which text editing completed
Resizing the chart or modifying the chart properties via the properties dialog window	CHARTEVENT_CHART_CHANGE	—	—	—
Custom event with N number	CHARTEVENT_CUSTOM+N	Value defined by the EventChartCustom() function	Value defined by the EventChartCustom() function	Value defined by the EventChartCustom() function

Sample chart event listener:

```
//+------------------------------------------------------------------+
//|                                         OnChartEvent_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property description "Sample chart event listener and custom events generator"
//--- service keys IDs
#define KEY_NUMPAD_5       12
#define KEY_LEFT           37
#define KEY_UP             38
#define KEY_RIGHT          39
#define KEY_DOWN           40
#define KEY_NUMLOCK_DOWN   98
#define KEY_NUMLOCK_LEFT  100
#define KEY_NUMLOCK_5     101
#define KEY_NUMLOCK_RIGHT 102
#define KEY_NUMLOCK_UP    104
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- display the CHARTEVENT_CUSTOM constant value
   Print("CHARTEVENT_CUSTOM=",CHARTEVENT_CUSTOM);
//---
   Print("Launched the EA ",MQLInfoString(MQL5_PROGRAM_NAME));
//--- set the flag of receiving chart object creation events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- set the flag of receiving chart object removal events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//--- enabling mouse wheel scrolling messages
   ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
//--- forced updating of chart properties ensures readiness for event processing
   ChartRedraw();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- tick counter for generating a custom event
   static int tick_counter=0;
//--- divide accumulated ticks by this value
   int simple_number=113;
//---
   tick_counter++;
//--- send a custom event if the tick counter is multiple of simple_number
   if(tick_counter%simple_number==0)
     {
      //--- form custom event ID from 0 to 65535
      ushort custom_event_id=ushort(tick_counter%65535);
      //---  send a custom event with parameters filling
      EventChartCustom(ChartID(),custom_event_id,tick_counter,SymbolInfoDouble(Symbol
      //--- add to a log for analyzing the example results
      Print(__FUNCTION__,": Sent a custom event ID=",custom_event_id);
     }
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- keypress
   if(id==CHARTEVENT_KEYDOWN)
     {
      switch((int)lparam)
        {
         case KEY_NUMLOCK_LEFT:  Print("Pressed KEY_NUMLOCK_LEFT");   break;
         case KEY_LEFT:          Print("Pressed KEY_LEFT");           break;
         case KEY_NUMLOCK_UP:    Print("Pressed KEY_NUMLOCK_UP");     break;
         case KEY_UP:            Print("Pressed KEY_UP");             break;
         case KEY_NUMLOCK_RIGHT: Print("Pressed KEY_NUMLOCK_RIGHT");  break;
         case KEY_RIGHT:         Print("Pressed KEY_RIGHT");          break;
         case KEY_NUMLOCK_DOWN:  Print("Pressed KEY_NUMLOCK_DOWN");   break;
         case KEY_DOWN:          Print("Pressed KEY_DOWN");           break;
         case KEY_NUMPAD_5:      Print("Pressed KEY_NUMPAD_5");       break;
         case KEY_NUMLOCK_5:     Print("Pressed KEY_NUMLOCK_5");      break;
         default:                Print("Pressed unlisted key");
        }
     }
//--- left-clicking on a chart
   if(id==CHARTEVENT_CLICK)
      Print("Mouse click coordinates on a chart: x = ",lparam,"  y = ",dparam);
//--- clicking on a graphical object
   if(id==CHARTEVENT_OBJECT_CLICK)
      Print("Clicking a mouse button on an object named '"+sparam+"'");
//--- object removed
   if(id==CHARTEVENT_OBJECT_DELETE)
      Print("Removed object named ",sparam);
//--- object created
   if(id==CHARTEVENT_OBJECT_CREATE)
      Print("Created object named ",sparam);
//--- changed object
   if(id==CHARTEVENT_OBJECT_CHANGE)
      Print("Changed object named ",sparam);
//--- object moved or anchor point coordinates changed
   if(id==CHARTEVENT_OBJECT_DRAG)
```

```
      Print("Changing anchor points of object named ",sparam);
//--- changed a text in the input field of the Edit graphical object
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
      Print("Changed text in Edit object ",sparam,"  id=",id);
//--- mouse movement events
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
   if(id==CHARTEVENT_MOUSE_WHEEL)
     {
      //--- Consider the state of mouse buttons and wheel for this event
      int flg_keys = (int)(lparam>>32);          // The flag of states of the Ctrl an
      int x_cursor = (int)(short)lparam;         // X coordinate where the mouse whee
      int y_cursor = (int)(short)(lparam>>16);   // Y coordinate where the mouse whee
      int delta    = (int)dparam;                // Total value of mouse scroll, trig
      //--- handling the flag
      string str_keys="";
      if((flg_keys&0x0001)!=0)
         str_keys+="LMOUSE ";
      if((flg_keys&0x0002)!=0)
         str_keys+="RMOUSE ";
      if((flg_keys&0x0004)!=0)
         str_keys+="SHIFT ";
      if((flg_keys&0x0008)!=0)
         str_keys+="CTRL ";
      if((flg_keys&0x0010)!=0)
         str_keys+="MMOUSE ";
      if((flg_keys&0x0020)!=0)
         str_keys+="X1MOUSE ";
      if((flg_keys&0x0040)!=0)
         str_keys+="X2MOUSE ";

      if(str_keys!="")
         str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1)+"'";
      PrintFormat("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x_
     }
//--- event of resizing the chart or modifying the chart properties using the properti
   if(id==CHARTEVENT_CHART_CHANGE)
      Print("Changing the chart size or properties");
//--- custom event
   if(id>CHARTEVENT_CUSTOM)
      PrintFormat("Custom event ID=%d, lparam=%d, dparam=%G, sparam=%s",id,lparam,dpar
  }
//+------------------------------------------------------------------+
//| MouseState                                                       |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // mouse left
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // mouse right
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // mouse middle
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // mouse first X key
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // mouse second X key
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // shift key
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // control key
   return(res);
  }
```

See also

EventChartCustom, Types of chart events, Event handling functions, Program running, Client terminal events

OnTester

The function is called in Expert Advisors when the Tester event occurs to perform necessary actions after testing.

```
double  OnTester(void);
```

Return Value

Value of the custom criterion optimization for assessing test results.

Note

The OnTester() function can be used only when testing EAs and is intended primarily for the calculation of a value that is used as a 'Custom max' criterion when optimizing input parameters.

During the genetic optimization, sorting results within one generation is performed in descending order. This means that the results with the highest value are deemed the best from the optimization criterion point of view. The worst values for such sorting are placed at the end and are subsequently discarded. Therefore, they do not take part in forming the next generation.

Thus, the OnTester() function allows you not only to create and save your own test results reports, but also control the optimization process to find the best parameters of the trading strategy.

Below is an **example** of calculating the custom criterion optimization. The idea is to calculate the linear regression of the balance graph. It is described in the article Optimizing a strategy using balance graph and comparing results with "Balance + max Sharpe Ratio" criterion.

```
//+------------------------------------------------------------------+
//|                                            OnTester_Sample.mq5 |
//|                          Copyright 2018, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA with the OnTester() handler"
#property description "As a custom optimization criterion, "
#property description "the ratio of the balance graph linear regression"
#property description "divided by the deviation mean-square error is returned"
//--- include the class for trading operations
#include <Trade\Trade.mqh>
//--- EA input parameters
input double Lots               = 0.1;     // Volume
input int    Slippage           = 10;      // Allowable slippage
input int    MovingPeriod       = 80;      // Moving average period
input int    MovingShift        = 6;       // Moving average shift
//--- global variables
int    IndicatorHandle=0;  // indicator handle
bool   IsHedging=false;    // flag of the account
CTrade trade;              // for performing trading operations
//---
```

```
#define EA_MAGIC 18052018
//+------------------------------------------------------------------+
//| Check for position opening conditions                            |
//+------------------------------------------------------------------+
void CheckForOpen(void)
  {
   MqlRates rt[2];
//--- trade only at the start of a new bar
   if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
     {
      Print("CopyRates of ",_Symbol," failed, no history");
      return;
     }
//--- tick volume
   if(rt[1].tick_volume>1)
      return;
//--- receive moving average values
   double   ma[1];
   if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
     {
      Print("CopyBuffer from iMA failed, no data");
      return;
     }
//--- check for a signal presence
   ENUM_ORDER_TYPE signal=WRONG_VALUE;
//--- candle opened higher but closed below the moving average
   if(rt[0].open>ma[0] && rt[0].close<ma[0])
      signal=ORDER_TYPE_BUY;    // buy signal
   else // candle opened lower but closed above the moving average
     {
      if(rt[0].open<ma[0] && rt[0].close>ma[0])
         signal=ORDER_TYPE_SELL;// sell signal
     }
//--- additional checks
   if(signal!=WRONG_VALUE)
     {
      if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
        {
         double price=SymbolInfoDouble(_Symbol,signal==ORDER_TYPE_SELL ? SYMBOL_BID:SY
         trade.PositionOpen(_Symbol,signal,Lots,price,0,0);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| Check for position closing conditions                            |
//+------------------------------------------------------------------+
void CheckForClose(void)
  {
   MqlRates rt[2];
//--- trade only at the start of a new bar
   if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
     {
      Print("CopyRates of ",_Symbol," failed, no history");
      return;
     }
   if(rt[1].tick_volume>1)
      return;
//--- receive moving average values
   double   ma[1];
   if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
     {
      Print("CopyBuffer from iMA failed, no data");
      return;
     }
//--- position has already been selected earlier using PositionSelect()
   bool signal=false;
   long type=PositionGetInteger(POSITION_TYPE);
//--- candle opened higher but closed below the moving average - close a short positi
   if(type==(long)POSITION_TYPE_SELL && rt[0].open>ma[0] && rt[0].close<ma[0])
      signal=true;
//--- candle opened lower but closed above the moving average - close a long position
   if(type==(long)POSITION_TYPE_BUY && rt[0].open<ma[0] && rt[0].close>ma[0])
      signal=true;
//--- additional checks
   if(signal)
     {
      if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
         trade.PositionClose(_Symbol,Slippage);
     }
//---
  }
//+-------------------------------------------------------------------+
//| Select a position considering an account type: Netting or Hedging |
//+-------------------------------------------------------------------+
bool SelectPosition()
  {
   bool res=false;
//--- select a position for a Hedging account
   if(IsHedging)
     {
      uint total=PositionsTotal();
      for(uint i=0; i<total; i++)
        {
         string position_symbol=PositionGetSymbol(i);
         if(_Symbol==position_symbol && EA_MAGIC==PositionGetInteger(POSITION_MAGIC))
           {
            res=true;
            break;
           }
        }
     }
//--- select a position for a Netting account
   else
     {
      if(!PositionSelect(_Symbol))
         return(false);
      else
         return(PositionGetInteger(POSITION_MAGIC)==EA_MAGIC); //---check Magic numbe
     }
//--- execution result
   return(res);
  }
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit(void)
  {
//--- set a trading type: Netting or Hedging
   IsHedging=((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)==ACCOU
//--- initialize an object for correct position control
   trade.SetExpertMagicNumber(EA_MAGIC);
   trade.SetMarginMode();
   trade.SetTypeFillingBySymbol(Symbol());
   trade.SetDeviationInPoints(Slippage);
//--- create Moving Average indicator
   IndicatorHandle=iMA(_Symbol,_Period,MovingPeriod,MovingShift,MODE_SMA,PRICE_CLOSE);
   if(IndicatorHandle==INVALID_HANDLE)
     {
      printf("Error creating iMA indicator");
      return(INIT_FAILED);
     }
//--- ok
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick(void)
  {
//--- if a position is already opened, check the closing condition
   if(SelectPosition())
      CheckForClose();
// check the position opening condition
   CheckForOpen();
//---
  }
```

```
//+------------------------------------------------------------------+
//| Tester function                                                  |
//+------------------------------------------------------------------+
double OnTester()
  {
//--- custom criterion optimization value (the higher, the better)
   double ret=0.0;
//--- get trade results to the array
   double array[];
   double trades_volume;
   GetTradeResultsToArray(array,trades_volume);
   int trades=ArraySize(array);
//--- if there are less than 10 trades, test yields no positive results
   if(trades<10)
      return (0);
//--- average result per trade
   double average_pl=0;
   for(int i=0;i<ArraySize(array);i++)
      average_pl+=array[i];
   average_pl/=trades;
//--- display the message for the single-test mode
   if(MQLInfoInteger(MQL_TESTER) && !MQLInfoInteger(MQL_OPTIMIZATION))
      PrintFormat("%s: Trades=%d, Average profit=%.2f",__FUNCTION__,trades,average_pl)
//--- calculate linear regression ratios for the profit graph
   double a,b,std_error;
   double chart[];
   if(!CalculateLinearRegression(array,chart,a,b))
      return (0);
//--- calculate the error of the chart deviation from the regression line
   if(!CalculateStdError(chart,a,b,std_error))
      return (0);
//--- calculate the ratio of trend profits to the standard deviation
   ret=(std_error == 0.0) ? a*trades : a*trades/std_error;
//--- return custom criterion optimization value
   return(ret);
  }
//+------------------------------------------------------------------+
//| Get the array of profits/losses from deals                       |
//+------------------------------------------------------------------+
bool GetTradeResultsToArray(double &pl_results[],double &volume)
  {
//--- request the complete trading history
   if(!HistorySelect(0,TimeCurrent()))
      return (false);
   uint total_deals=HistoryDealsTotal();
   volume=0;
//--- set the initial size of the array with a margin - by the number of deals in hist
   ArrayResize(pl_results,total_deals);
//--- counter of deals that fix the trading result - profit or loss
```

```
   int counter=0;
   ulong ticket_history_deal=0;
//--- go through all deals
   for(uint i=0;i<total_deals;i++)
     {
      //--- select a deal
      if((ticket_history_deal=HistoryDealGetTicket(i))>0)
        {
         ENUM_DEAL_ENTRY deal_entry  =(ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket_hi
         long            deal_type   =HistoryDealGetInteger(ticket_history_deal,DEAL_T
         double          deal_profit =HistoryDealGetDouble(ticket_history_deal,DEAL_P
         double          deal_volume =HistoryDealGetDouble(ticket_history_deal,DEAL_V
         //--- we are only interested in trading operations
         if((deal_type!=DEAL_TYPE_BUY) && (deal_type!=DEAL_TYPE_SELL))
            continue;
         //--- only deals that fix profits/losses
         if(deal_entry!=DEAL_ENTRY_IN)
           {
            //--- write the trading result to the array and increase the counter of de
            pl_results[counter]=deal_profit;
            volume+=deal_volume;
            counter++;
           }
        }
     }
//--- set the final size of the array
   ArrayResize(pl_results,counter);
   return (true);
  }
//+------------------------------------------------------------------+
//| Calculate the linear regression y=a*x+b                          |
//+------------------------------------------------------------------+
bool CalculateLinearRegression(double  &change[],double &chartline[],
                               double  &a_coef,double  &b_coef)
  {
//--- check for data sufficiency
   if(ArraySize(change)<3)
      return (false);
//--- create a chart array with an accumulation
   int N=ArraySize(change);
   ArrayResize(chartline,N);
   chartline[0]=change[0];
   for(int i=1;i<N;i++)
      chartline[i]=chartline[i-1]+change[i];
//--- now, calculate regression ratios
   double x=0,y=0,x2=0,xy=0;
   for(int i=0;i<N;i++)
     {
      x=x+i;
      y=y+chartline[i];
      xy=xy+i*chartline[i];
      x2=x2+i*i;
     }
   a_coef=(N*xy-x*y)/(N*x2-x*x);
   b_coef=(y-a_coef*x)/N;
//---
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculate mean-square deviation error for specified a and b     |
//+------------------------------------------------------------------+
bool  CalculateStdError(double  &data[],double  a_coef,double  b_coef,double &std_err)
  {
//--- sum of error squares
   double error=0;
   int N=ArraySize(data);
   if(N<=2)
      return (false);
   for(int i=0;i<N;i++)
      error+=MathPow(a_coef*i+b_coef-data[i],2);
   std_err=MathSqrt(error/(N-2));
//---
   return (true);
  }
```

See also

Testing trading strategies, TesterHideIndicators, Working with optimization results, TesterStatistics, OnTesterInit, OnTesterDeinit, OnTesterPass, MQL_TESTER, MQL_OPTIMIZATION, FileOpen, FileWrite, FileLoad, FileSave

OnTesterInit

The function is called in EAs when the TesterInit event occurs to perform necessary actions before optimization in the strategy tester. There are two function types.

The **version that returns the result**

```
int  OnTesterInit(void);
```

Return Value

int type value, zero means successful initialization of an EA launched on a chart before optimization starts.

The OnTesterInit() call that returns the execution result is recommended for use since it not only allows for program initialization, but also returns an error code in case of an early optimization stop. Return of any value other than INIT_SUCCEEDED (0) means an error, no optimization is launched.

The version without a result return is left only for compatibility with old codes. Not recommended for use

```
void  OnTesterInit(void);
```

Note

The TesterInit event is generated before EA optimization in the strategy tester starts. At this event, an EA having OnTesterDeInit() or OnTesterPass() event handler is automatically downloaded on a separate terminal chart. It has the symbol and the period that have been specified in the tester.

Such an event receives the TesterInit, TesterDeinit and TesterPass events, but not Init, Deinit and NewTick ones. Accordingly, all necessary logic for processing the results of each pass during optimization should be implemented in the OnTesterInit (), OnTesterDeinit () and OnTesterPass () handlers.

The result of each single pass during a strategy optimization can be passed via a frame from the OnTester () handler using the FrameAdd () function.

The OnTesterInit() function is used to initiate an Expert Advisor before start of optimization for further processing of optimization results. It is always used together with the OnTesterDeinit() handler.

The time for OnTesterInit() execution is limited. If it is exceeded, the EA is forcibly stopped, while the optimization itself is canceled. A message is displayed in the tester journal:

```
TesterOnTesterInit works too long. Tester cannot be initialized.
```

The example is taken from OnTick. The OnTesterInit() handler is added for setting optimization parameters:

```
//+------------------------------------------------------------------+
//|                                         OnTesterInit_Sample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Sample EA with the OnTesterInit() handler,"
#property description "in which values and limitations of "
#property description "inputs during optimization are set"

input double lots=0. 1;       // volume in lots
input double kATR=3;          // signal candle length in ATR
input int    ATRperiod=20;    // ATR indicator period
input int    holdbars=8;      // number of bars to hold position on
input int    slippage=10;     // allowable slippage
input bool   revers=false;    // reverse the signal?
input ulong  EXPERT_MAGIC=0;  // EA's MagicNumber
//--- for storing the ATR indicator handle
int atr_handle;
//--- here we will store the last ATR values and the candle body
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
//--- remember optimization start time
datetime optimization_start;
//--- for displaying duration on a chart after the end of optimization
string report;
//+------------------------------------------------------------------+
//| TesterInit function                                              |
//+------------------------------------------------------------------+
void OnTesterInit()
  {
//--- set the values of inputs for optimization
   ParameterSetRange("lots",false,0.1,0,0,0);
   ParameterSetRange("kATR",true,3.0,1.0,0.3,7.0);
   ParameterSetRange("ATRperiod",true,10,15,1,30);
   ParameterSetRange("holdbars",true,5,3,1,15);
   ParameterSetRange("slippage",false,10,0,0,0);
   ParameterSetRange("revers",true,false,false,1,true);
   ParameterSetRange("EXPERT_MAGIC",false,123456,0,0,0);
   Print("Initial values and optimization parameter limitations are set");
//--- remember optimization start
   optimization_start=TimeLocal();
   report=StringFormat("%s: optimization launched at %s",
                       __FUNCTION__,TimeToString(TimeLocal(),TIME_MINUTES|TIME_SECONDS
//--- show messages on the chart and the terminal journal
   Print(report);
   Comment(report);
//---
  }
//+------------------------------------------------------------------+
//| TesterDeinit function                                            |
//+------------------------------------------------------------------+
void OnTesterDeinit()
  {
//--- optimization duration
   string log_message=StringFormat("%s: optimization took %d seconds",
                                   __FUNCTION__,TimeLocal()-optimization_start);
   PrintFormat(log_message);
   report=report+"\r\n"+log_message;
   Comment(report);
  }
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- initialize global variables
   last_atr=0;
   last_body=0;
//--- set the correct volume
   double min_lot=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   trade_lot=lots>min_lot? lots:min_lot;
//--- create ATR indicator handle
   atr_handle=iATR(_Symbol,_Period,ATRperiod);
   if(atr_handle==INVALID_HANDLE)
     {
      PrintFormat("%s: failed to create iATR, error code %d",__FUNCTION__,GetLastErro
      return(INIT_FAILED);
     }
//--- successful EA initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- trading signal
   static int signal=0; // +1 means a buy signal, -1 means a sell signal
//--- check and close old positions opened more than 'holdbars' bars ago
   ClosePositionsByBars(holdbars,slippage,EXPERT_MAGIC);
//--- check for a new bar
   if(isNewBar())
     {
      //--- check for a signal presence
      signal=CheckSignal();
     }
//--- if a netting position is opened, skip the signal - wait till it closes
   if(signal!=0 && PositionsTotal()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
     {
      signal=0;
      return; // exit the NewTick event handler and do not enter the market before a
     }
//--- for a hedging account, each position is held and closed separately
   if(signal!=0)
     {
      //--- buy signal
      if(signal>0)
        {
         PrintFormat("%s: Buy signal! Revers=%s",__FUNCTION__,string(revers));
         if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
      //--- sell signal
      if(signal<0)
        {
         PrintFormat("%s: Sell signal! Revers=%s",__FUNCTION__,string(revers));
         if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
        }
     }
//--- OnTick function end
  }
//+------------------------------------------------------------------+
//| Check for a new trading signal                                   |
//+------------------------------------------------------------------+
int CheckSignal()
  {
//--- 0 means no signal
   int res=0;
//--- get ATR value on a penultimate complete bar (the bar index is 2)
   double atr_value[1];
   if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
     {
      last_atr=atr_value[0];
      //--- get data on the last closed bar to the MqlRates type array
      MqlRates bar[1];
      if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
        {
         //--- calculate the bar body size on the last complete bar
         last_body=bar[0].close-bar[0].open;
         //--- if the body of the last bar (with index 1) exceeds the previous ATR val
         if(MathAbs(last_body)>kATR*last_atr)
            res=last_body>0?1:-1; // positive value for the upward candle
        }
      else
         PrintFormat("%s: Failed to receive the last bar! Error",__FUNCTION__,GetLast
     }
   else
```

```
      PrintFormat("%s: Failed to receive ATR indicator value! Error",__FUNCTION__,Get
//--- if reverse trading mode is enabled
   res=revers?-res:res;  // reverse the signal if necessary (return -1 instead of 1 a
//--- return a trading signal value
   return (res);
  }
//+------------------------------------------------------------------+
//|  Return 'true' when a new bar appears                            |
//+------------------------------------------------------------------+
bool isNewBar(const bool print_log=true)
  {
   static datetime bartime=0; // store open time of the current bar
//--- get open time of the zero bar
   datetime currbar_time=iTime(_Symbol,_Period,0);
//--- if open time changes, a new bar has arrived
   if(bartime!=currbar_time)
     {
      bartime=currbar_time;
      lastbar_timeopen=bartime;
      //--- display data on open time of a new bar in the log
      if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION)||MQLInfoInteger(MQL_TESTER)))
        {
         //--- display a message with a new bar open time
         PrintFormat("%s: new bar on %s %s opened at %s",__FUNCTION__,_Symbol,
                     StringSubstr(EnumToString(_Period),7),
                     TimeToString(TimeCurrent(),TIME_SECONDS));
         //--- get data on the last tick
         MqlTick last_tick;
         if(!SymbolInfoTick(Symbol(),last_tick))
            Print("SymbolInfoTick() failed, error = ",GetLastError());
         //--- display the last tick time up to milliseconds
         PrintFormat("Last tick was at %s.%03d",
                     TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%1000
        }
      //--- we have a new bar
      return (true);
     }
//--- no new bar
   return (false);
  }
//+------------------------------------------------------------------+
//| Buy at a market price with a specified volume                    |
//+------------------------------------------------------------------+
bool Buy(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- buy at a market price
   return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| Sell at a market price with a specified volume                   |
//+------------------------------------------------------------------+
bool Sell(double volume,ulong deviation=10,ulong  magicnumber=0)
  {
//--- sell at a market price
   return (MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber));
  }
//+------------------------------------------------------------------+
//| Close positions by hold time in bars                             |
//+------------------------------------------------------------------+
void ClosePositionsByBars(int holdtimebars,ulong deviation=10,ulong  magicnumber=0)
  {
   int total=PositionsTotal(); // number of open positions
//--- iterate over open positions
   for(int i=total-1; i>=0; i--)
     {
      //--- position parameters
      ulong  position_ticket=PositionGetTicket(i);
      string position_symbol=PositionGetString(POSITION_SYMBOL);
      ulong  magic=PositionGetInteger(POSITION_MAGIC);
      datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
      int bars=iBarShift(_Symbol,PERIOD_CURRENT,position_open)+1;

      //--- if a position's lifetime is already large, while MagicNumber and a symbol
      if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
         int    digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
         double volume=PositionGetDouble(POSITION_VOLUME);
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         string str_type=StringSubstr(EnumToString(type),14);
         StringToLower(str_type); // lower the text case for correct message formattin
         PrintFormat("Close position #%I64u %s %s %.2f",
                     position_ticket,position_symbol,str_type,volume);
         //--- set an order type and sending a trade request
         if(type==POSITION_TYPE_BUY)
            MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber,position_ticket);
         else
            MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber,position_ticket);
        }
     }
  }
//+------------------------------------------------------------------+
//| Prepare and send a trade request                                 |
//+------------------------------------------------------------------+
bool MarketOrder(ENUM_ORDER_TYPE type,double volume,ulong slip,ulong magicnumber,ulong
  {
//--- declaring and initializing structures
   MqlTradeRequest request={};
   MqlTradeResult  result={};
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(type==ORDER_TYPE_BUY)
      price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- request parameters
   request.action    =TRADE_ACTION_DEAL;                     // trading operation type
   request.position  =pos_ticket;                            // position ticket if clo
   request.symbol    =Symbol();                              // symbol
   request.volume    =volume;                                // volume
   request.type      =type;                                  // order type
   request.price     =price;                                 // trade price
   request.deviation=slip;                                   // allowable deviation fr
   request.magic     =magicnumber;                           // order MagicNumber
//--- send a request
   if(!OrderSend(request,result))
     {
      //--- display data on failure
      PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
                  request.symbol,EnumToString(type),volume,request.price,GetLastError
      return (false);
     }
//--- inform of a successful operation
   PrintFormat("retcode=%u  deal=%I64u  order=%I64u",result.retcode,result.deal,result
   return (true);
  }
```

See also

[Testing trading strategies](), [Working with optimization results](), [OnTesterDeinit](), [OnTesterPass](), [ParameterGetRange](), [ParameterSetRange]()

OnTesterDeinit

The function is called in EAs when the TesterDeinit event occurs after EA optimization.

```
void  OnTesterDeinit(void);
```

Return Value

No return value

Note

The TesterDeinit event is generated after the end of EA optimization in the strategy tester.

An EA having OnTesterDeInit() or OnTesterPass() event handler is automatically downloaded on a separate terminal chart during the optimization start. It has the symbol and the period that have been specified in the tester. The function is designed for the final processing of all optimization results.

Keep in mind that optimization frames sent by test agents using the FrameAdd() function may come in bundles and take time to deliver. Therefore, not all frames, as well as TesterPass events, may arrive and be processed in OnTesterPass() before the end of optimization. If you want to receive all belated frames in OnTesterDeinit(), place the code block using the FrameNext() function.

See also

Testing trading strategies, Working with optimization results, TesterStatistics, OnTesterInit, OnTesterPass, ParameterGetRange, ParameterSetRange

OnTesterPass

The function is called in EAs when the TesterPass event occurs for handling a new data frame during EA optimization.

```
void  OnTesterPass(void);
```

Return Value

No return value

Note

The TesterPass event is generated automatically when receiving a frame during an Expert Advisor optimization in the strategy tester.

An EA having OnTesterDeInit() or OnTesterPass() event handler is automatically downloaded on a separate terminal chart during the optimization start. It has the symbol and the period that have been specified in the tester. The function is meant for handling frames received from test agents during optimization. The frame containing test results should be sent from the OnTester() handler using the FrameAdd() function.

Keep in mind that optimization frames sent by test agents using the FrameAdd() function may come in bundles and take time to deliver. Therefore, not all frames, as well as TesterPass events, may arrive and be processed in OnTesterPass() before the end of optimization. If you want to receive all belated frames in OnTesterDeinit(), place the code block using the FrameNext() function.

After completing OnTesterDeinit() optimization, it is possible to sort all received frames again using the FrameFirst()/FrameFilter and FrameNext() functions.

See also

Testing trading strategies, Working with optimization results, OnTesterInit, OnTesterDeinit, FrameFirst, FrameFilter, FrameNext, FrameInputs

Getting Market Information

These are functions intended for receiving information about the market state.

Function	Action
SymbolsTotal	Returns the number of available (selected in Market Watch or all) symbols
SymbolExist	Checks if a symbol with a specified name exists
SymbolName	Returns the name of a specified symbol
SymbolSelect	Selects a symbol in the Market Watch window or removes a symbol from the window
SymbolIsSynchronized	Checks whether data of a selected symbol in the terminal are synchronized with data on the trade server
SymbolInfoDouble	Returns the double value of the symbol for the corresponding property
SymbolInfoInteger	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property
SymbolInfoString	Returns a value of the string type of a specified symbol for the corresponding property
SymbolInfoMarginRate	Returns the margin rates depending on the order type and direction
SymbolInfoTick	Returns the current prices for the specified symbol in a variable of the MqlTick type
SymbolInfoSessionQuote	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.
SymbolInfoSessionTrade	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.
MarketBookAdd	Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes
MarketBookRelease	Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes
MarketBookGet	Returns a structure array MqlBookInfo containing records of the Depth of Market of a specified symbol

SymbolsTotal

Returns the number of available (selected in Market Watch or all) symbols.

```
int  SymbolsTotal(
   bool  selected      // True - only symbols in MarketWatch
   );
```

Parameters

selected

[in] Request mode. Can be true or false.

Return Value

If the 'selected' parameter is true, the function returns the number of symbols selected in MarketWatch. If the value is false, it returns the total number of all symbols.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of available symbols
   int total    = SymbolsTotal(false);        // all symbols on the server
   int selected = SymbolsTotal(true);         // all symbols added to the Market Watch

//--- send the obtained data to the journal
   PrintFormat("Number of available symbols on the server: %d\n"+
               "Number of available symbols selected in MarketWatch: %d", total, selec
   /*
   result:
   Number of available symbols on the server: 140
   Number of available symbols selected in MarketWatch: 11
   */
  }
```

SymbolExist

Checks if a symbol with a specified name exists.

```
bool  SymbolExist(
   const string  name,     // symbol name
   bool&   is_custom       // custom symbol property
   );
```

Parameters

name

[in] Symbol name.

is_custom

[out] Custom symbol property set upon successful execution. If true, the detected symbol is a custom one.

Return Value

If false, the symbol is not found among standard and custom ones.

Example:

```
#define SYMBOL_NAME "GBPUSDn"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the custom symbol flag and check the presence of a symbol having its na
   bool custom = false;
   bool result = SymbolExist(SYMBOL_NAME, custom);

//--- declare the default 'symbol not found' message text
   string text = StringFormat("The symbol '%s' was not found among either the standar

//--- if a symbol is found, create a message text depending on which list the symbol
   if(result)
     {
      //--- if this is a standard symbol
      if(!custom)
         text = StringFormat("The '%s' symbol is available on the server.", SYMBOL_NA
      //--- if this is a custom symbol
      else
         text = StringFormat("The symbol '%s' was found in the list of custom symbols
     }

//--- send the message about the check result to the journal
   Print(text);
   /*
   result for standard 'GBPUSD' symbol:
   The 'GBPUSD' symbol is available on the server.

   result for custom 'GBPUSDx' symbol:
   The symbol 'GBPUSDx' was found in the list of custom symbols.

   result for missing 'GBPUSDn' symbol:
   The symbol 'GBPUSDn' was not found among either the standard or custom symbols.
   */
  }
```

See also

SymbolsTotal, SymbolSelect, Custom symbols

SymbolName

Returns the name of a symbol.

```
string  SymbolName(
   int    pos,          // number in the list
   bool   selected      // true - only symbols in MarketWatch
   );
```

Parameters

pos

[in] Order number of a symbol.

selected

[in] Request mode. If the value is true, the symbol is taken from the list of symbols selected in MarketWatch. If the value is false, the symbol is taken from the general list.

Return Value

Value of string type with the symbol name.

Example:

```
#define SYMBOL_NAME "GBPHKD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the flag of the server symbol search result
   bool found = false;

//--- find the 'SYMBOL_NAME' symbol in the list of all server symbols
   int total = SymbolsTotal(false);
   for(int i=0; i<total; i++)
     {
      //--- get a symbol name in the list by loop index
      string name = SymbolName(i, false);

      //--- if this is the desired symbol, send its name and position in the list to t
      if(name == SYMBOL_NAME)
        {
         PrintFormat("The '%s' symbol was found in the list of server symbols at index
         found = true;
         break;
        }
     }

//--- if a symbol is not found on the server, report this before shutting down
```

```
    if(!found)
       PrintFormat("The '%s' symbol was not found on the server.", SYMBOL_NAME);
   }
```

SymbolSelect

Selects a symbol in the Market Watch window or removes a symbol from the window.

```
bool  SymbolSelect(
   string  name,        // symbol name
   bool    select       // add or remove
   );
```

Parameters

name

[in] Symbol name.

select

[in] Switch. If the value is false, a symbol should be removed from MarketWatch, otherwise a symbol should be selected in this window. A symbol can't be removed if the symbol chart is open, or there are open positions for this symbol.

Return Value

In case of failure returns false.

Example:

```
#define SYMBOL_NAME "GBPHKD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check for the presence of a symbol in the lists, if not found, report it and co
   bool custom = false;
   if(!SymbolExist(SYMBOL_NAME, custom))
     {
      PrintFormat("'%s' symbol not found in the lists", SYMBOL_NAME);
      return;
     }

//--- add a symbol to the Market Watch window
   ResetLastError();
   if(!SymbolSelect(SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }
//--- if a symbol is successfully added to the list, get its index in the Market Watc
   int index = SymbolIndex(SYMBOL_NAME);
   PrintFormat("The '%s' symbol has been added to the MarketWatch list. Symbol index 
//--- now remove the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(SYMBOL_NAME, false))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }
//--- if a symbol is successfully removed from the list, its index in the Market Watc
   index = SymbolIndex(SYMBOL_NAME);
   PrintFormat("The '%s' symbol has been removed from the MarketWatch list. Symbol in

   /*
   result:
   The 'GBPHKD' symbol has been added to the MarketWatch list. Symbol index in the li
   The 'GBPHKD' symbol has been removed from the MarketWatch list. Symbol index in the
   */
  }
//+------------------------------------------------------------------+
//| Return the symbol index in the Market Watch symbol list          |
//+------------------------------------------------------------------+
int SymbolIndex(const string symbol)
  {
   int total = SymbolsTotal(true);
   for(int i=0; i<total; i++)
     {
      string name = SymbolName(i, true);
      if(name == symbol)
         return i;
     }
   return(WRONG_VALUE);
  }
```

SymbolIsSynchronized

The function checks whether data of a selected symbol in the terminal are synchronized with data on the trade server.

```
bool  SymbolIsSynchronized(
   string  name,       // symbol name
   );
```

Parameters

name

[in] Symbol name.

Return value

If data are synchronized, returns 'true'; otherwise returns 'false'.

Example:

```
#define SYMBOL_NAME "EURUSD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the flag for synchronizing data in the terminal using the 'SYMBOL_NAME' symb
   bool   sync = SymbolIsSynchronized(SYMBOL_NAME);

//--- create a message depending on the synchronization flag
   string text = StringFormat("The data on the '%s' symbol in the terminal is synchron
   if(!sync)
      text = StringFormat("The data for the '%s' symbol in the terminal is not synchro

//--- send the obtained result to the journal
   Print(text);

   /*
   result for synchronized data:
   The data on the 'EURUSD' symbol in the terminal is synchronized with the data on th

   result for data not yet synchronized:
   The data for the 'GBPHKD' symbol in the terminal is not synchronized with the data
   */
  }
```

See also

SymbolInfoInteger, Organizing Data Access

SymbolInfoDouble

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  SymbolInfoDouble(
   string                     name,        // symbol
   ENUM_SYMBOL_INFO_DOUBLE    prop_id      // identifier of the property
   );
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool  SymbolInfoDouble(
   string                     name,        // symbol
   ENUM_SYMBOL_INFO_DOUBLE    prop_id,     // identifier of the property
   double&                    double_var   // here we accept the property value
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_DOUBLE enumeration.

double_var

[out] Variable of double type receiving the value of the requested property.

Return Value

The value of double type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 5040 – invalid string parameter for specifying a symbol name,
- 4301 – unknown symbol (financial instrument),
- 4302 – symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 – invalid identifier of a symbol property.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

The SymbolInfoMarginRate() function provides data on the amount of charged margin depending on the order type and direction.

Example:

```
void OnTick()
  {
//--- obtain spread from the symbol properties
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- now let's calculate the spread by ourselves
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

SymbolInfoInteger

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  SymbolInfoInteger(
   string                      name,      // symbol
   ENUM_SYMBOL_INFO_INTEGER  prop_id    // identifier of a property


   );
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool  SymbolInfoInteger(
   string                      name,      // symbol
   ENUM_SYMBOL_INFO_INTEGER  prop_id,   // identifier of a property
   long&                       long_var   // here we accept the property value
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_INTEGER enumeration.

long_var

[out] Variable of the long type receiving the value of the requested property.

Return Value

The value of long type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 5040 – invalid string parameter for specifying a symbol name,
- 4301 – unknown symbol (financial instrument),
- 4302 – symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 – invalid identifier of a symbol property.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:

```
void OnTick()
  {
//--- obtain spread from the symbol properties
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- now let's calculate the spread by ourselves
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

SymbolInfoString

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  SymbolInfoString(
   string                   name,         // Symbol
   ENUM_SYMBOL_INFO_STRING  prop_id       // Property identifier
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a placeholder variable passed by reference in the last parameter.

```
bool  SymbolInfoString(
   string                   name,         // Symbol
   ENUM_SYMBOL_INFO_STRING  prop_id,      // Property identifier
   string&                  string_var    // here we accept the property value
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_STRING enumeration.

string_var

[out] Variable of the string type receiving the value of the requested property.

Return Value

The value of string type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 5040 – invalid string parameter for specifying a symbol name,
- 4301 – unknown symbol (financial instrument),
- 4302 – symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4303 – invalid identifier of a symbol property.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:

```
#define SYMBOL_NAME "USDJPY"
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get string data for SYMBOL_NAME symbol
   string currency_base   = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_BASE);    //
   string currency_profit = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_PROFIT);  //
   string currency_margin = SymbolInfoString(SYMBOL_NAME, SYMBOL_CURRENCY_MARGIN);  //
   string symbol_descript = SymbolInfoString(SYMBOL_NAME, SYMBOL_DESCRIPTION);      //

//--- create a message text with the obtained data
   string text=StringFormat("Symbol %s:\n"+
                            "- Currency Base: %s\n"+
                            "- Currensy Profit: %s\n"+
                            "- Currency Margin: %s\n"+
                            "- Symbol Description: %s",
                            SYMBOL_NAME, currency_base,
                            currency_profit, currency_margin,
                            symbol_descript);

//--- send the obtained data to the journal
   Print(text);
   /*
   Symbol USDJPY:
   - Currency Base: USD
   - Currensy Profit: JPY
   - Currency Margin: USD
   - Symbol Description: US Dollar vs Yen
   */
  }
```

SymbolInfoMarginRate

Returns the margin rates depending on the order type and direction.

```
bool  SymbolInfoMarginRate(
   string              name,                     // symbol name
   ENUM_ORDER_TYPE     order_type,               // order type
   double&             initial_margin_rate,      // initial margin rate
   double&             maintenance_margin_rate   // maintenance margin rate
   );
```

Parameters

name

[in] Symbol name.

order_type

[in] Order type.

initial_margin_rate

[in] A double type variable for receiving an initial margin rate. Initial margin is a security deposit for 1 lot deal in the appropriate direction. Multiplying the rate by the initial margin, we receive the amount of funds to be reserved on the account when placing an order of the specified type.

maintenance_margin_rate

[out] A double type variable for receiving a maintenance margin rate. Maintenance margin is a minimum amount for maintaining an open position of 1 lot in the appropriate direction. Multiplying the rate by the maintenance margin, we receive the amount of funds to be reserved on the account after an order of the specified type is activated.

Return Value

Returns true if request for properties is successful, otherwise false.

Example:

```
#define SYMBOL_NAME Symbol()

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the variables the ratios are set into
   double initial_margin_rate = 0;      // initial margin rate
   double maintenance_margin_rate = 0; // maintenance margin rate

//--- get and send the SYMBOL_NAME symbol ratios for the Buy market order to the jour
   SymbolInfoMarginRatePrint(SYMBOL_NAME, ORDER_TYPE_BUY, initial_margin_rate, mainte

//--- get and send the SYMBOL_NAME symbol ratios for the Sell market order to the jou
   SymbolInfoMarginRatePrint(SYMBOL_NAME, ORDER_TYPE_SELL, initial_margin_rate, maint
   /*
   result:
   Initial margin rate for symbol AFKS for the Buy market order: 0.225600
   Maintenance margin rate for symbol AFKS for the Buy market order: 0.112800
   Initial margin rate for symbol AFKS for the Sell market order: 0.254400
   Maintenance margin rate for symbol AFKS for the Sell market order: 0.127200
   */
  }
//+------------------------------------------------------------------+
//| Send the margin ratios to the journal                            |
//+------------------------------------------------------------------+
void SymbolInfoMarginRatePrint(const string symbol, const ENUM_ORDER_TYPE order_type,
  {
//--- get the 'symbol' ratios for the order_type market order
   ResetLastError();
   if(!SymbolInfoMarginRate(symbol, order_type, initial_margin_rate, maintenance_marg
     {
      Print("SymbolInfoMarginRate() failed. Error ", GetLastError());
      return;
     }
//--- print the obtained ratio values
   string type=(order_type==ORDER_TYPE_BUY ? "Buy" : order_type==ORDER_TYPE_SELL ? "Se
   PrintFormat("Initial margin rate for symbol %s for the %s market order: %f\n"+
               "Maintenance margin rate for symbol %s for the %s market order: %f",
               SYMBOL_NAME, type, initial_margin_rate,
               SYMBOL_NAME, type, maintenance_margin_rate);
  }
```

SymbolInfoTick

The function returns current prices of a specified symbol in a variable of the MqlTick type.

```
bool  SymbolInfoTick(
   string     symbol,      // symbol name
   MqlTick&   tick         // reference to a structure
   );
```

Parameters

symbol

[in] Symbol name.

tick

[out] Link to the structure of the MqlTick type, to which the current prices and time of the last price update will be placed.

Return Value

The function returns true if successful, otherwise returns false.

Example:

```
#define SYMBOL_NAME "EURUSD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare an array with the MqlTick structure type of dimension 1
   MqlTick tick[1]={};

//--- get the latest prices for the SYMBOL_NAME symbol into the MqlTick structure
   if(!SymbolInfoTick(SYMBOL_NAME, tick[0]))
     {
      Print("SymbolInfoTick() failed. Error ", GetLastError());
      return;
     }

//--- send the obtained data to the journal
   PrintFormat("Latest price data for the '%s' symbol:", SYMBOL_NAME);
   ArrayPrint(tick);
   /*
   result:
   Latest price data for the 'EURUSD' symbol:
                    [time]   [bid]   [ask] [last] [volume]    [time_msc] [flags] [volu
   [0] 2024.05.17 23:58:54 1.08685 1.08695 0.0000        0 1715990334319       6
   */
  }
```

SymbolInfoSessionQuote

Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.

```
bool  SymbolInfoSessionQuote(
   string            name,                // symbol name
   ENUM_DAY_OF_WEEK  day_of_week,         // day of the week
   uint              session_index,       // session index
   datetime&         from,                // time of the session beginning
   datetime&         to                   // time of the session end
   );
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration ENUM_DAY_OF_WEEK.

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return Value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

Example:

```
#define SYMBOL_NAME   Symbol()
#define SESSION_INDEX 0

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the header with a symbol and SESSION_INDEX and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of t
   PrintFormat("Symbol %s, Quote session %d:", SYMBOL_NAME, SESSION_INDEX);
   for(int i=MONDAY; i<SATURDAY; i++)
```

```
      SymbolInfoSessionQuotePrint(SYMBOL_NAME, (ENUM_DAY_OF_WEEK)i, SESSION_INDEX);
   /*
   result:
   Symbol RU000A103661, Quote session 0:
   - Monday     06:45 - 00:00
   - Tuesday    06:45 - 00:00
   - Wednesday  06:45 - 00:00
   - Thursday   06:45 - 00:00
   - Friday     06:45 - 00:00
   */
  }
//+------------------------------------------------------------------+
//| Send the start and end times of the specified quote session      |
//| for the specified symbol and day of the week to the journal      |
//+------------------------------------------------------------------+
void SymbolInfoSessionQuotePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- declare variables to record the beginning and end of the quote session
   datetime date_from;  // session start time
   datetime date_to;    // session end time

//--- get data from the quotation session by symbol and day of the week
   if(!SymbolInfoSessionQuote(symbol, day_of_week, session_index, date_from, date_to))
     {
      Print("SymbolInfoSessionQuote() failed. Error ", GetLastError());
      return;
     }

//--- create the week day name from the enumeration constant
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- send data for the specified quote session to the journal
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Ti
  }
```

See also

Symbol Properties, TimeToStruct, Data Structures

SymbolInfoSessionTrade

Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.

```
bool  SymbolInfoSessionTrade(
   string            name,                // symbol name
   ENUM_DAY_OF_WEEK  day_of_week,         // day of the week
   uint              session_index,       // session index
   datetime&         from,                // session beginning time
   datetime&         to                   // session end time
   );
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration ENUM_DAY_OF_WEEK.

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

Example:

```
#define SYMBOL_NAME   Symbol()
#define SESSION_INDEX 0

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the header with a symbol and SESSION_INDEX and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of t
   PrintFormat("Symbol %s, Trade session %d:", SYMBOL_NAME, SESSION_INDEX);
   for(int i=MONDAY; i<SATURDAY; i++)
```

```
      SymbolInfoSessionTradePrint(SYMBOL_NAME, (ENUM_DAY_OF_WEEK)i, SESSION_INDEX);
   /*
   result:
   Symbol GBPUSD, Trade session 0:
   - Monday     00:15 - 23:55
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   */
  }
//+------------------------------------------------------------------+
//| Send the start and end times of the specified trade session      |
//| for the specified symbol and day of the week to the journal      |
//+------------------------------------------------------------------+
void SymbolInfoSessionTradePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- declare variables to record the beginning and end of the trading session
   datetime date_from;  // session start time
   datetime date_to;    // session end time

//--- get data from the trading session by symbol and day of the week
   if(!SymbolInfoSessionTrade(symbol, day_of_week, session_index, date_from, date_to))
     {
      Print("SymbolInfoSessionTrade() failed. Error ", GetLastError());
      return;
     }

//--- create the week day name from the enumeration constant
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- send data for the specified trading session to the journal
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Tir
  }
```

See also

Symbol Properties, TimeToStruct, Data Structures

MarketBookAdd

Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes.

```
bool  MarketBookAdd(
   string  symbol      // symbol
   );
```

Parameters

symbol

[in] The name of a symbol, whose Depth of Market is to be used in the Expert Advisor or script.

Return Value

The true value if opened successfully, otherwise false.

Note

Normally, this function must be called from the OnInit() function or in the class constructor. To handle incoming alerts, in the Expert Advisor program must contain the function void OnBookEvent(string& symbol).

Example:

```
#define   SYMBOL_NAME   "GBPUSD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the market depth for SYMBOL_NAME symbol
   if(!MarketBookAdd(SYMBOL_NAME))
     {
      PrintFormat("MarketBookAdd(%s) failed. Error ", SYMBOL_NAME, GetLastError());
      return;
     }

//--- send the message about successfully opening the market depth to the journal
   PrintFormat("The MarketBook for the '%s' symbol was successfully opened and a subs

//--- wait 2 seconds
   Sleep(2000);

//--- upon completion, unsubscribe from the open market depth
   ResetLastError();
   if(MarketBookRelease(SYMBOL_NAME))
      PrintFormat("MarketBook for the '%s' symbol was successfully closed", SYMBOL_NA
   else
      PrintFormat("Error %d occurred when closing MarketBook using the '%s' symbol", 
  /*
  result:
  The MarketBook for the 'GBPUSD' symbol was successfully opened and a subscription t
  MarketBook for the 'GBPUSD' symbol was successfully closed
  */
 }
```

See also

Structure of Depth of Market, Structures and Classes

MarketBookRelease

Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes.

```
bool  MarketBookRelease(
   string  symbol      // symbol
   );
```

Parameters

symbol

[in] Symbol name.

Return Value

The true value if closed successfully, otherwise false.

Note

Normally, this function must be called from the OnDeinit() function, if the corresponding MarketBookAdd() function has been called in the OnInit() function. Or it must be called from the class destructor, if the corresponding MarketBookAdd() function has been called from the class constructor.

Example:

```
#define SYMBOL_NAME "GBPUSD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the market depth for SYMBOL_NAME symbol
   if(!MarketBookAdd(SYMBOL_NAME))
     {
      PrintFormat("MarketBookAdd(%s) failed. Error ", SYMBOL_NAME, GetLastError());
      return;
     }

//--- send the message about successfully opening the market depth to the journal
   PrintFormat("The MarketBook for the '%s' symbol was successfully opened and a subsc

//--- wait 2 seconds
   Sleep(2000);

//--- upon completion, unsubscribe from the open market depth
//--- send the message about successfully unsubscribing from the market depth or about
   ResetLastError();
   if(MarketBookRelease(SYMBOL_NAME))
      PrintFormat("MarketBook for the '%s' symbol was successfully closed", SYMBOL_NAM
   else
      PrintFormat("Error %d occurred when closing MarketBook using the '%s' symbol", 

   /*
   result:
   The MarketBook for the 'GBPUSD' symbol was successfully opened and a subscription 
   MarketBook for the 'GBPUSD' symbol was successfully closed
   */
  }
```

See also

Structure of Depth of Market, Structures and Classes

MarketBookGet

Returns a structure array MqlBookInfo containing records of the Depth of Market of a specified symbol.

```
bool  MarketBookGet(
   string         symbol,     // symbol
   MqlBookInfo&   book[]      // reference to an array
   );
```

Parameters

symbol

[in] Symbol name.

book[]

[in] Reference to an array of Depth of Market records. The array can be pre-allocated for a sufficient number of records. If a dynamic array hasn't been pre-allocated in the operating memory, the client terminal will distribute the array itself.

Return Value

Returns true in case of success, otherwise false.

Note

The Depth of Market must be pre-opened by the MarketBookAdd() function.

Example:

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo for ",Symbol());
      for(int i=0;i<size;i++)
        {
         Print(i+":",priceArray[i].price
               +"    Volume = "+priceArray[i].volume,
               " type = ",priceArray[i].type);
        }
     }
   else
     {
      Print("Could not get contents of the symbol DOM ",Symbol());
     }
```

See also

Structure of Depth of Market, Structures and Classes

Economic calendar functions

This section describes the functions for working with the economic calendar available directly in the MetaTrader platform. The economic calendar is a ready-made encyclopedia featuring descriptions of macroeconomic indicators, their release dates and degrees of importance. Relevant values of macroeconomic indicators are sent to the MetaTrader platform right at the moment of publication and are displayed on a chart as tags allowing you to visually track the required indicators by countries, currencies and importance.

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

Economic calendar functions allow conducting the auto analysis of incoming events according to custom importance criteria from a perspective of necessary countries/currencies.

Function	Action
CalendarCountryById	Get a country description by its ID
CalendarEventById	Get an event description by its ID
CalendarValueById	Get an event value description by its ID
CalendarCountries	Get the array of country names available in the calendar
CalendarEventByCountry	Get the array of descriptions of all events available in the calendar by a specified country code
CalendarEventByCurrency	Get the array of descriptions of all events available in the calendar by a specified currency
CalendarValueHistoryByEvent	Get the array of values for all events in a specified time range by an event ID
CalendarValueHistory	Get the array of values for all events in a specified time range with the ability to sort by country and/or currency
CalendarValueLastByEvent	Get the array of event values by its ID since the calendar database status with a specified change_id
CalendarValueLast	Get the array of values for all events with the ability to sort by country and/or currency since the calendar database status with a specified change_id

CalendarCountryById

Get a country description by its ID.

```
bool  CalendarCountryById(
   const long             country_id,     // country ID
   MqlCalendarCountry&    country         // variable for receiving a country descriptio
   );
```

Parameters

country_id

[in] Country ID ([ISO 3166-1](https://en.wikipedia.org/wiki/ISO_3166-1)).

country

[out] [MqlCalendarCountry](#) type variable for receiving a country description.

Return Value

Returns true if successful, otherwise - false. To get information about an error, call the [GetLastError()](#) function. Possible errors:

- 4001 - ERR_INTERNAL_ERROR (general runtime error),
- 5402 - ERR_CALENDAR_NO_DATA (country is not found),
- 5401 - ERR_CALENDAR_TIMEOUT (request time limit exceeded).

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the list of countries from the economic calendar
   MqlCalendarCountry countries[];
   int count=CalendarCountries(countries);
//--- check the result
   if(count==0)
      PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
//--- if there are two or more countries
   if(count>=2)
     {
      MqlCalendarCountry country;
      //--- now get a country description by its ID
      if(CalendarCountryById(countries[1].id, country))
        {
         //--- prepare a country description
         string descr="id = "+IntegerToString(country.id)+"\n";
         descr+=("name = " + country.name+"\n");
         descr+=("code = " + country.code+"\n");
         descr+=("currency = " + country.currency+"\n");
         descr+=("currency_symbol = " + country.currency_symbol+"\n");
```

```
          descr+=("url_name = " + country.url_name);
          //--- display a country description
          Print(descr);
         }
       else
          Print("CalendarCountryById() failed. Error ",GetLastError());
      }
//---
  }
/*
  Result:
   id = 999
   name = European Union
   code = EU
   currency = EUR
   currency_symbol = €
   url_name = european-union
*/
```

See also

[CalendarCountries](), [CalendarEventByCountry]()

CalendarEventById

Get an event description by its ID.

```
bool  CalendarEventById(
   ulong                event_id,      // event ID
   MqlCalendarEvent&    event          // variable for receiving an event description
   );
```

Parameters

event_id

[in] Event ID.

event

[out] MqlCalendarEvent type variable for receiving an event description.

Return Value

Returns true if successful, otherwise - false. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 5402 – ERR_CALENDAR_NO_DATA (country is not found),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded).

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- country code for Germany (ISO 3166-1 Alpha-2)
   string germany_code="DE";
//--- get Germany events
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(germany_code,events);
//--- display Germany events in the Journal
   if(events_count>0)
     {
      PrintFormat("Germany events: %d",events_count);
      ArrayPrint(events);
     }
   else
     {
      PrintFormat("Failed to receive events for the country code %s, error %d",
                  germany_code,GetLastError());
      //--- script early completion
      return;
     }
//--- get description of the last event from the events[] array
```

```
   MqlCalendarEvent event;
   ulong event_id=events[events_count-1].id;
   if(CalendarEventById(event_id,event))
     {
      MqlCalendarCountry country;
      CalendarCountryById(event.country_id,country);
      PrintFormat("Event description with event_id=%d received",event_id);
      PrintFormat("Country: %s (country code = %d)",country.name,event.country_id);
      PrintFormat("Event name: %s",event.name);
      PrintFormat("Event code: %s",event.event_code);
      PrintFormat("Event importance: %s",EnumToString((ENUM_CALENDAR_EVENT_IMPORTANCE)
      PrintFormat("Event type: %s",EnumToString((ENUM_CALENDAR_EVENT_TYPE)event.type))
      PrintFormat("Event sector: %s",EnumToString((ENUM_CALENDAR_EVENT_SECTOR)event.se
      PrintFormat("Event frequency: %s",EnumToString((ENUM_CALENDAR_EVENT_FREQUENCY)ev
      PrintFormat("Event release mode: %s",EnumToString((ENUM_CALENDAR_EVENT_TIMEMODE)
      PrintFormat("Event measurement unit: %s",EnumToString((ENUM_CALENDAR_EVENT_UNIT)
      PrintFormat("Number of decimal places: %d",event.digits);
      PrintFormat("Event multiplier: %s",EnumToString((ENUM_CALENDAR_EVENT_MULTIPLIER)
      PrintFormat("Source URL: %s",event.source_url);
     }
   else
      PrintFormat("Failed to get event description for event_d=%s, error %d",
                  event_id,GetLastError());
  }
/*
  Result:
  Germany events: 50
             [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importa
   [ 0] 276010001      1        6           2           0          276      1
   [ 1] 276010002      1        6           2           0          276      1
   [ 2] 276010003      1        4           2           0          276      1
   [ 3] 276010004      1        4           2           0          276      1
   ....
   [47] 276500001      1        8           2           0          276      0
   [48] 276500002      1        8           2           0          276      0
   [49] 276500003      1        8           2           0          276      0
  Event description with event_id=276500003 received
  Country: Germany (country code = 276)
  Event name: Markit Composite PMI
  Event code: markit-composite-pmi
   Event importance: CALENDAR_IMPORTANCE_MODERATE
   Event type: CALENDAR_TYPE_INDICATOR
   Event sector: CALENDAR_SECTOR_BUSINESS
   Event frequency: CALENDAR_FREQUENCY_MONTH
   Event release mode: CALENDAR_TIMEMODE_DATETIME
   Event measurement unit: CALENDAR_UNIT_NONE
  Number of decimal places: 1
   Value multiplier: CALENDAR_MULTIPLIER_NONE
   Source URL: https://www.markiteconomics.com
```

```
*/
```

See also

CalendarEventByCountry, CalendarEventByCurrency, CalendarValueById

CalendarValueById

Get an event value description by its ID.

```
bool  CalendarValueById(
   ulong                 value_id,      // event value ID
   MqlCalendarValue&     value          // variable for receiving an event value
   );
```

Parameters

value_id

[in] Event value ID.

value

[out] MqlCalendarValue type variable for receiving an event description. See the example of handling calendar events.

Return Value

Returns true if successful, otherwise - false. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 5402 – ERR_CALENDAR_NO_DATA (country is not found),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded).

Note

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

The MqlCalendarValue structure provides methods for checking and setting values from the actual_value, forecast_value, prev_value and revised_prev_value fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in a field, then you should divide the value by 1,000,000 in order to get the value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- country code for Japan (ISO 3166-1 Alpha-2)
```

```
   string japan_code="JP";
//--- set the boundaries of the interval we take the events from
   datetime date_from=D'01.01.2018';  // take all events from 2018
   datetime date_to=0;                // 0 means all known events, including the ones
//--- get the array of the Japan event values
   MqlCalendarValue values[];
   int values_count=CalendarValueHistory(values,date_from,date_to,japan_code);
//--- move along the detected event values
   if(values_count>0)
     {
      PrintFormat("Number of values for Japan events: %d",values_count);
      //--- delete all "empty" values (actual_value==-9223372036854775808)
      for(int i=values_count-1;i>=0;i--)
        {
         if(values[i].actual_value==-9223372036854775808)
            ArrayRemove(values,i,1);
        }
      PrintFormat("Number of values after deleting empty ones: %d",ArraySize(values));
     }
   else
     {
      PrintFormat("Failed to receive events for the country code %s, error %d",
                  japan_code,GetLastError());
      //--- script early completion
      return;
     }
//--- leave no more than 10 values in the values[] array
   if(ArraySize(values)>10)
     {
      PrintFormat("Reduce the list of values to 10 and display them");
      ArrayRemove(values,0,ArraySize(values)-10);
     }
   ArrayPrint(values);

//--- now let's display how to get an event value description based on the known value
   for(int i=0;i<ArraySize(values);i++)
     {
      MqlCalendarValue value;
      CalendarValueById(values[i].id,value);
      PrintFormat("%d: value_id=%d value=%d impact=%s",
                  i,values[i].id,value.actual_value,EnumToString(ENUM_CALENDAR_EVENT_
     }
//---
  }
/*
  Result:
  Number of values for Japan events: 1734
  Number of values after deleting empty ones: 1017
  Reduce the list of values to 10 and display them
```

```
        [id] [event_id]              [time]            [period] [revision] [actual_val
   [0] 56500  392030004 2019.03.28 23:30:00 2019.03.01 00:00:00          0         900
   [1] 56501  392030005 2019.03.28 23:30:00 2019.03.01 00:00:00          0         700
   [2] 56502  392030006 2019.03.28 23:30:00 2019.03.01 00:00:00          0        1100
   [3] 56544  392030007 2019.03.28 23:30:00 2019.02.01 00:00:00          0        2300
   [4] 56556  392050002 2019.03.28 23:30:00 2019.02.01 00:00:00          0        1630
   [5] 55887  392020003 2019.03.28 23:50:00 2019.02.01 00:00:00          0         400
   [6] 55888  392020004 2019.03.28 23:50:00 2019.02.01 00:00:00          0       -1800
   [7] 55889  392020002 2019.03.28 23:50:00 2019.02.01 00:00:00          0         200
   [8] 55948  392020006 2019.03.28 23:50:00 2019.02.01 00:00:00          1        1400
   [9] 55949  392020007 2019.03.28 23:50:00 2019.02.01 00:00:00          1       -1000
  Display brief data on event values based on value_id
   0: value_id=56500 value=900000 impact=CALENDAR_IMPACT_POSITIVE
   1: value_id=56501 value=700000 impact=CALENDAR_IMPACT_NA
   2: value_id=56502 value=1100000 impact=CALENDAR_IMPACT_POSITIVE
   3: value_id=56544 value=2300000 impact=CALENDAR_IMPACT_NEGATIVE
   4: value_id=56556 value=1630000 impact=CALENDAR_IMPACT_POSITIVE
   5: value_id=55887 value=400000 impact=CALENDAR_IMPACT_NEGATIVE
   6: value_id=55888 value=-1800000 impact=CALENDAR_IMPACT_POSITIVE
   7: value_id=55889 value=200000 impact=CALENDAR_IMPACT_NEGATIVE
   8: value_id=55948 value=1400000 impact=CALENDAR_IMPACT_POSITIVE
   9: value_id=55949 value=-1000000 impact=CALENDAR_IMPACT_NEGATIVE
*/
```

See also

CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent, CalendarValueLast

CalendarCountries

Get the array of country names available in the Calendar.

```
int  CalendarCountries(
   MqlCalendarCountry&  countries[]         // array for receiving a list of Calendar
   );
```

Parameters

countries[]

[out] An array of MqlCalendarCountry type for receiving all Calendar countries' descriptions.

Return Value

Number of received descriptions. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- 5400 – ERR_CALENDAR_MORE_DATA (array size is insufficient for receiving descriptions of all countries, only the ones that managed to fit in were received).

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the list of countries from the economic calendar
   MqlCalendarCountry countries[];
   int count=CalendarCountries(countries);
//--- display the array in the Journal
   if(count>0)
      ArrayPrint(countries);
   else
      PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
/*
  Result:
        [id]           [name] [code] [currency] [currency_symbol]       [url_name] [re
    [ 0]     0 "Worldwide"      "WW"   "ALL"      ""                 "worldwide"
    [ 1]   999 "European Union" "EU"   "EUR"      "€"                "european-union"
    [ 2]   840 "United States"  "US"   "USD"      "$"                "united-states"
    [ 3]   124 "Canada"         "CA"   "CAD"      "$"                "canada"
    [ 4]    36 "Australia"      "AU"   "AUD"      "$"                "australia"
    [ 5]   554 "New Zealand"    "NZ"   "NZD"      "$"                "new-zealand"
    [ 6]   392 "Japan"          "JP"   "JPY"      "I"                "japan"
    [ 7]   156 "China"          "CN"   "CNY"      "I"                "china"
    [ 8]   826 "United Kingdom" "GB"   "GBP"      "J"                "united-kingdom"
    [ 9]   756 "Switzerland"    "CH"   "CHF"      "₣"                "switzerland"
    [10]  276 "Germany"         "DE"   "EUR"      "€"                "germany"
    [11]  250 "France"          "FR"   "EUR"      "€"                "france"
    [12]  380 "Italy"           "IT"   "EUR"      "€"                "italy"
    [13]  724 "Spain"           "ES"   "EUR"      "€"                "spain"
    [14]   76 "Brazil"          "BR"   "BRL"      "R$"               "brazil"
    [15]  410 "South Korea"     "KR"   "KRW"      "₩"                "south-korea"
*/
  }
```

See also

CalendarEventByCountry, CalendarCountryById

CalendarEventByCountry

Get the array of descriptions of all events available in the Calendar by a specified country code.

```
int  CalendarEventByCountry(
   string                country_code,      // country code name (ISO 3166-1 alpha-2)
   MqlCalendarEvent&     events[]           // variable for receiving the description a
   );
```

Parameters

country_code

[in] Country code name (ISO 3166-1 alpha-2)

events[]

[out] MqlCalendarEvent type array for receiving descriptions of all events for a specified country.

Return Value

Number of received descriptions. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- errors of failed execution of ArrayResize()

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- country code for EU (ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- get EU events
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(EU_code,events);
//--- display EU events in the Journal
   if(events_count>0)
     {
      PrintFormat("EU events: %d",events_count);
      ArrayPrint(events);
     }
//---
  }
/*
  Result:
  EU events: 56
               [id] [type]  [country_id] [unit] [importance] [multiplier] [digits] [ever
    [ 0] 999010001      0           999      0            2            0        0  "ECB
    [ 1] 999010002       0           999      0             2             0        0  "ECB
    [ 2] 999010003       0           999      0             3             0        0  "ECB
    [ 3] 999010004       0           999      0             3             0        0  "ECB
    [ 4] 999010005       0           999      0             2             0        0  "ECB
    [ 5] 999010006       1           999      1             3             0        2  "ECB
    [ 6] 999010007       1           999      1             3             0        2  "ECB
    [ 7] 999010008       0           999      0             2             0        0  "ECB
    [ 8] 999010009       1           999      2             2             3        3  "ECB
    [ 9] 999010010       0           999      0             2             0        0  "ECB
    [10] 999010011       0           999      0             2             0        0  "ECB
    ...

*/
```

See also

[CalendarCountries](), [CalendarCountryById]()

CalendarEventByCurrency

Get the array of descriptions of all events available in the Calendar by a specified currency.

```
int  CalendarEventByCurrency(
   const string          currency,     // country currency code name
   MqlCalendarEvent&     events[]      // variable for receiving the description array
   );
```

Parameters

currency

[in] Country currency code name.

events[]

[out] MqlCalendarEvent type array for receiving descriptions of all events for a specified currency.

Return Value

Number of received descriptions. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- errors of failed execution of ArrayResize()

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare the array for receiving economic calendar events
      MqlCalendarEvent events[];
//--- get EU currency events
      int count = CalendarEventByCurrency("EUR",events);
      Print("count = ", count);
//--- 10 events are sufficient for the current example
      if(count>10)
         ArrayResize(events,10);
//--- display events in the Journal
      ArrayPrint(events);
  }
/*
  Result:
             [id] [type] [country_id] [unit] [importance]
   [0] 999010001      0          999      0            2 "https://www.ecb.europa.eu/h
   [1] 999010002      0          999      0            2 "https://www.ecb.europa.eu/h
   [2] 999010003      0          999      0            3 "https://www.ecb.europa.eu/h
```

```
    [3] 999010004      0          999      0            3 "https://www.ecb.europa.eu/h
    [4] 999010005      0          999      0            2 "https://www.ecb.europa.eu/h
    [5] 999010006      1          999      1            3 "https://www.ecb.europa.eu/h
    [6] 999010007      1          999      1            3 "https://www.ecb.europa.eu/h
    [7] 999010008      0          999      0            2 "https://www.ecb.europa.eu/h
    [8] 999010009      1          999      2            2 "https://www.ecb.europa.eu/h
    [9] 999010010      0          999      0            2 "https://www.ecb.europa.eu/h
*/
```

See also

CalendarEventById, CalendarEventByCountry

CalendarValueHistoryByEvent

Get the array of values for all events in a specified time range by an event ID.

```
int  CalendarValueHistoryByEvent(
   ulong              event_id,          // event ID
   MqlCalendarValue&  values[],          // array for value descriptions
   datetime           datetime_from,     // left border of a time range
   datetime           datetime_to=0      // right border of a time range
   );
```

Parameters

event_id

[in] Event ID.

values[]

[out] MqlCalendarValue type array for receiving event values. See the example of handling calendar events.

datetime_from

[in] Initial date of a time range events are selected from by a specified ID, while *datetime_from* < *datetime_to*.

datetime_to=0

[in] End date of a time range events are selected from by a specified ID. If the *datetime_to* is not set (or is 0), all event values beginning from the specified *datetime_from* date in the Calendar database are returned (including the values of future events).

Return Value

If successful, return the number of available values in the 'values' array, otherwise -1. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- 5400 – ERR_CALENDAR_MORE_DATA (array size is insufficient for receiving descriptions of all values, only the ones that managed to fit in were received),
- errors of failed execution of ArrayResize()

The MqlCalendarValue structure provides methods for checking and setting values from the actual_value, forecast_value, prev_value and revised_prev_value fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in a field, then you should divide the value by 1,000,000 in order to get the value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

Note

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- country code for EU (ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- get EU events
   MqlCalendarEvent events[];
   int events_count=CalendarEventByCountry(EU_code,events);
//--- display EU events in the Journal
   if(events_count>0)
     {
      PrintFormat("EU events: %d",events_count);
      //--- reduce the event list, 10 events are sufficient for analysis
      ArrayResize(events,10);
      ArrayPrint(events);
     }
//--- see that the "ECB Interest Rate Decision" event has event_id=999010007
   ulong event_id=events[6].id;        // the event's ID may change in the Calendar,
   string event_name=events[6].name;   // name of a Calendar event
   PrintFormat("Get values for event_name=%s event_id=%d",event_name,event_id);
//--- get all values of the "ECB Interest Rate Decision" event
   MqlCalendarValue values[];
//--- set the boundaries of the interval we take the events from
   datetime date_from=0;           // take all events from the beginning of the availa
   datetime date_to=D'01.01.2016'; // take events not older than 2016
   if(CalendarValueHistoryByEvent(event_id,values,date_from,date_to))
     {
      PrintFormat("Received values for %s: %d",
                  event_name,ArraySize(values));
      //--- reduce the value list, 10 events are sufficient for analysis
      ArrayResize(values,10);
      ArrayPrint(values);
     }
   else
     {
      PrintFormat("Error! Failed to get values for event_id=%d",event_id);
      PrintFormat("Error code: %d",GetLastError());
     }
  }
//---
```

```
/*
  Result:
  EU events: 56
            [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importar
   [0] 999010001      0        5           0           0          999      0
   [1] 999010002      0        5           0           0          999      0
   [2] 999010003      0        5           0           0          999      0
   [3] 999010004      0        5           0           0          999      0
   [4] 999010005      0        5           0           0          999      0
   [5] 999010006      1        5           0           0          999      1
   [6] 999010007      1        5           0           0          999      1
   [7] 999010008      0        5           0           0          999      0
   [8] 999010009      1        5           0           0          999      2
   [9] 999010010      0        5           0           0          999      0
  Get values for event_name=ECB Interest Rate Decision event_id=999010007
  Received ECB Interest Rate Decision event values: 102
       [id] [event_id]              [time]            [period] [revision] [actual_valu
   [0] 2776  999010007 2007.03.08 11:45:00 1970.01.01 00:00:00          0        3750
   [1] 2777  999010007 2007.05.10 11:45:00 1970.01.01 00:00:00          0        3750
   [2] 2778  999010007 2007.06.06 11:45:00 1970.01.01 00:00:00          0        4000
   [3] 2779  999010007 2007.07.05 11:45:00 1970.01.01 00:00:00          0        4000
   [4] 2780  999010007 2007.08.02 11:45:00 1970.01.01 00:00:00          0        4000
   [5] 2781  999010007 2007.09.06 11:45:00 1970.01.01 00:00:00          0        4000
   [6] 2782  999010007 2007.10.04 11:45:00 1970.01.01 00:00:00          0        4000
   [7] 2783  999010007 2007.11.08 12:45:00 1970.01.01 00:00:00          0        4000
   [8] 2784  999010007 2007.12.06 12:45:00 1970.01.01 00:00:00          0        4000
   [9] 2785  999010007 2008.01.10 12:45:00 1970.01.01 00:00:00          0        4000
*/
```

See also

CalendarCountries, CalendarEventByCountry, CalendarValueHistory, CalendarEventById, CalendarValueById

CalendarValueHistory

Get the array of values for all events in a specified time range with the ability to sort by country and/or currency.

```
int  CalendarValueHistory(
   MqlCalendarValue&  values[],              // array for value descriptions
   datetime           datetime_from,         // left border of a time range
   datetime           datetime_to=0          // right border of a time range
   const string       country_code=NULL,     // country code name (ISO 3166-1 alpha-2)
   const string       currency=NULL          // country currency code name
   );
```

Parameters

values[]

[out] MqlCalendarValue type array for receiving event values. See the example of handling calendar events.

datetime_from

[in] Initial date of a time range events are selected from by a specified ID, while *datetime_from* < *datetime_to*.

datetime_to=0

[in] End date of a time range events are selected from by a specified ID. If the *datetime_to* is not set (or is 0), all event values beginning from the specified *datetime_from* date in the Calendar database are returned (including the values of future events).

country_code=NULL

[in] Country code name (ISO 3166-1 alpha-2)

currency=NULL

[in] Country currency code name.

Return Value

If successful, return the number of available values in the 'values' array, otherwise -1. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- 5400 – ERR_CALENDAR_MORE_DATA (array size is insufficient for receiving descriptions of all values, only the ones that managed to fit in were received),
- errors of failed execution of ArrayResize()

Note

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

If the *events[]* array of fixed length was passed to the function and there was not enough space to save the entire result, the ERR_CALENDAR_MORE_DATA (5400) error is activated.

If the *datetime_to* is not set (or is 0), all event values beginning from the specified *datetime_from* date in the Calendar database are returned (including the values of future events).

For the *country_code* and *currency* filters, NULL and "" values are equivalent and mean the absence of the filter.

For *country_code*, the *code* field of the MqlCalendarCountry structure, for example "US", "RU" or "EU", should be used.

For *currency*, the *currency* field of the MqlCalendarCountry structure, for example "USD", "RUB" or "EUR", should be used.

The filters are applied by conjunction, i.e. logical 'AND' is used to select only the values of events both conditions (country and currency) are simultaneously met for.

The MqlCalendarValue structure provides methods for checking and setting values from the actual_value, forecast_value, prev_value and revised_prev_value fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in a field, then you should divide the value by 1,000,000 in order to get the value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- country code for EU (ISO 3166-1 Alpha-2)
   string EU_code="EU";
//--- get all EU event values
   MqlCalendarValue values[];
//--- set the boundaries of the interval we take the events from
   datetime date_from=D'01.01.2018';  // take all events from 2018
   datetime date_to=0;                // 0 means all known events, including the ones
//--- request EU event history since 2018 year
   if(CalendarValueHistory(values,date_from,date_to,EU_code))
     {
      PrintFormat("Received event values for country_code=%s: %d",
                  EU_code,ArraySize(values));
      //--- decrease the size of the array for outputting to the Journal
      ArrayResize(values,10);
//--- display event values in the Journal
      ArrayPrint(values);
     }
   else
     {
      PrintFormat("Error! Failed to receive events for country_code=%s",EU_code);
      PrintFormat("Error code: %d",GetLastError());
     }
//---
  }
/*
  Result:
  Received event values for country_code=EU: 1384
        [id] [event_id]           [time]               [period] [revision]   [actual_v
   [0] 54215  999500001 2018.01.02 09:00:00 2017.12.01 00:00:00          3       6060
   [1] 54221  999500002 2018.01.04 09:00:00 2017.12.01 00:00:00          3       5660
   [2] 54222  999500003 2018.01.04 09:00:00 2017.12.01 00:00:00          3       5810
   [3] 45123  999030005 2018.01.05 10:00:00 2017.11.01 00:00:00          0         60
   [4] 45124  999030006 2018.01.05 10:00:00 2017.11.01 00:00:00          0        280
   [5] 45125  999030012 2018.01.05 10:00:00 2017.12.01 00:00:00          1         90
   [6] 45126  999030013 2018.01.05 10:00:00 2017.12.01 00:00:00          1        140
   [7] 54953  999520001 2018.01.05 20:30:00 2018.01.02 00:00:00          0      12790
   [8] 22230  999040003 2018.01.08 10:00:00 2017.12.01 00:00:00          0        910
   [9] 22231  999040004 2018.01.08 10:00:00 2017.12.01 00:00:00          0       1840
*/
```

See also

CalendarCountries, CalendarEventByCountry, CalendarValueHistoryByEvent, CalendarEventById, CalendarValueById

CalendarValueLastByEvent

Get the array of event values by its ID since the Calendar database status with a specified change_id.

```
int  CalendarValueLastByEvent(
   ulong                event_id,      // event ID
   ulong&               change_id,     // Calendar change ID
   MqlCalendarValue&    values[]       // array for value descriptions
   );
```

Parameters

event_id

[in] Event ID.

change_id

[in][out] Change ID.

values[]

[out] MqlCalendarValue type array for receiving event values. See the example of handling calendar events.

Return Value

Number of received event values. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- 5400 – ERR_CALENDAR_MORE_DATA (array size is insufficient for receiving descriptions of all values, only the ones that managed to fit in were received),
- errors of failed execution of ArrayResize()

Note

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

If the *events[]* array of fixed length was passed to the function and there was not enough space to save the entire result, the ERR_CALENDAR_MORE_DATA (5400) error is activated.

If *change_id* = 0 is passed to the function, the function always returns zero but the current calendar database is returned to *change_id*.

The function returns the array for a specified news and a new *change_id* that can be used for subsequent calls of the function to receive the new values of the news. Thus, it is possible to update values for a specified news by calling this function with the last known *change_id*.

The MqlCalendarValue structure provides methods for checking and setting values from the actual_value, forecast_value, prev_value and revised_prev_value fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in a field, then you should divide the value by 1,000,000 in order to get the value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

The sample EA listening for the Nonfarm payrolls report release:

```
#property description "Example of using the CalendarValueLastByEvent function"
#property description " for tracking the release of the Nonfarm Payrolls report."
#property description "To achieve this, get the current change ID"
#property description " of the Calendar database. Then, use this ID to receive"
#property description " only new events via the timer survey"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create timer
   EventSetTimer(60);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy timer
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- Calendar database change ID
   static ulong calendar_change_id=0;
//--- first launch attribute
   static bool first=true;
```

```
//--- event ID
   static ulong event_id=0;
//--- event name
   static string event_name=NULL;
//--- event value array
   MqlCalendarValue values[];
//--- perform initialization - get the current calendar_change_id
   if(first)
     {
      MqlCalendarEvent events[];
      //--- country code for USA (ISO 3166-1 Alpha-2)
      string USA_code="US";
      //--- get events for USA
      int events_count=CalendarEventByCountry(USA_code,events);
      //--- position of a necessary event in the 'events' array
      int event_pos=-1;
      //--- display USA events in the Journal
      if(events_count>0)
        {
         PrintFormat("%s: USA events: %d",__FUNCTION__,events_count);
         for(int i=0;i<events_count;i++)
           {
            string event_name_low=events[i].name;
            //--- change an event name to lower case
            if(!StringToLower(event_name_low))
              {
               PrintFormat("StringToLower() returned %d error",GetLastError());
               //--- exit the function ahead of time
               return;
              }
            //--- look for the "Nonfarm Payrolls" event
            if(StringFind(event_name_low,"nonfarm payrolls")!=-1)
              {
               //--- event found, remember its ID
               event_id=events[i].id;
               //--- write the "Nonfarm Payrolls" event name
               event_name=events[i].name;
               //--- remember the events' position in the 'events[]' array
               event_pos=i;
               //--- keep in mind that the Calendar features several events containin
               PrintFormat("Event \"Nonfarm Payrolls\" found: event_id=%d  event_name=
               //--- view all the events by commenting out the 'break' operator to bet
               break;
              }
           }
         //--- reduce the list by deleting events after "Nonfarm Payrolls"
         ArrayRemove(events,event_pos+1);
         //--- leave 9 events before "Nonfarm Payrolls" for more convenient analysis
         ArrayRemove(events,0,event_pos-9);
         ArrayPrint(events);
        }
      else
        {
         PrintFormat("%s: CalendarEventByCountry(%s) returned 0 events, error code=%d"
                     USA_code,__FUNCTION__,GetLastError());
         //--- operation completed in a failure, try again during the next call of the
         return;
        }

      //--- get the Calendar database change ID for the specified event
      if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
        {
         //--- this code block cannot be executed during the first launch but let's ac
         PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                     __FUNCTION__,calendar_change_id);
         //--- set the flag and exit before the timer's next event
         first=false;
         return;
        }
      else
        {
         //--- data are not received (this is normal for the first launch), check for
         int error_code=GetLastError();
         if(error_code==0)
           {
            PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                        __FUNCTION__,calendar_change_id);
            //--- set the flag and exit before the timer's next event
            first=false;
            //--- now we have the calendar_change_id value
            return;
           }
         else
           {
            //--- and this is really an error
            PrintFormat("%s: Failed to get values for event_id=%d",__FUNCTION__,event_
            PrintFormat("Error code: %d",error_code);
            //--- operation completed in a failure, try again during the next call of
            return;
           }
        }
     }

//--- we have the last known value of the Calendar change ID (change_id)
   ulong old_change_id=calendar_change_id;
//--- check for a new Nonfarm Payrolls event value
   if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
     {
      PrintFormat("%s: Received new events for \"%s\": %d",
                  __FUNCTION__,event_name,ArraySize(values));
      //--- display data from the 'values' array in the Journal
      ArrayPrint(values);
      //--- display the values of the previous and new Calendar IDs in the Journal
      PrintFormat("%s: Previous change_id=%d, new change_id=%d",
                  __FUNCTION__,old_change_id,calendar_change_id);
/*
      write your code that is to handle "Nonfarm Payrolls" data release here
      */
     }
//---
  }
/*
  Result:
   OnTimer: USA events: 202
  Event "Nonfarm Payrolls" found: event_id=840030016  event_name=Nonfarm Payrolls
            [id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importar
   [0] 840030007      1        4           2           0          840      1
   [1] 840030008      1        4           2           0          840      1
   [2] 840030009      1        4           2           0          840      0
   [3] 840030010      1        4           2           0          840      0
   [4] 840030011      1        4           2           0          840      1
   [5] 840030012      1        4           2           0          840      1
   [6] 840030013      1        4           2           0          840      1
   [7] 840030014      1        4           2           0          840      1
   [8] 840030015      1        3           2           0          840      1
   [9] 840030016      1        3           2           0          840      4
   OnTimer: Received the Calendar database current ID: change_id=33986560

*/
```

See also

[CalendarValueLast](), [CalendarValueHistory](), [CalendarValueHistoryByEvent](), [CalendarValueById]()

CalendarValueLast

Get the array of values for all events with the ability to sort by country and/or currency since the calendar database status with a specified change_id.

```
int  CalendarValueLast(
   ulong&               change_id,            // change ID
   MqlCalendarValue&    values[],             // array for value descriptions
   const string         country_code=NULL,    // country code name (ISO 3166-1 alpha-
   const string         currency=NULL         // country currency code name
   );
```

Parameters

change_id

[in][out] Change ID.

values[]

[out] MqlCalendarValue type array for receiving event values. See the example of handling calendar events.

country_code=NULL

[in] Country code name (ISO 3166-1 alpha-2)

currency=NULL

[in] Country currency code name.

Return Value

Number of received event values. To get information about an error, call the GetLastError() function. Possible errors:

- 4001 – ERR_INTERNAL_ERROR (general runtime error),
- 4004 – ERR_NOT_ENOUGH_MEMORY (not enough memory for executing a request),
- 5401 – ERR_CALENDAR_TIMEOUT (request time limit exceeded),
- 5400 – ERR_CALENDAR_MORE_DATA (array size is insufficient for receiving descriptions of all values, only the ones that managed to fit in were received),
- errors of failed execution of ArrayResize()

Note

All functions for working with the economic calendar use the trade server time (TimeTradeServer). This means that the time in the MqlCalendarValue structure and the time inputs in the CalendarValueHistoryByEvent/CalendarValueHistory functions are set in a trade server timezone, rather than a user's local time.

If the *events[]* array of fixed length was passed to the function and there was not enough space to save the entire result, the ERR_CALENDAR_MORE_DATA (5400) error is activated.

If *change_id* = 0 is passed to the function, you will get the current *change_id* of the calendar database to that parameter; and the function returns 0

For the *country_code* and *currency* filters, NULL and "" values are equivalent and mean the absence of the filter.

For *country_code*, the *code* field of the MqlCalendarCountry structure, for example "US", "RU" or "EU", should be used.

For *currency*, the *currency* field of the MqlCalendarCountry structure, for example "USD", "RUB" or "EUR", should be used.

The filters are applied by conjunction, i.e. logical 'AND' is used to select only the values of events both conditions (country and currency) are simultaneously met for

The function returns the array for a specified news and a new *change_id* that can be used for subsequent calls of the function to receive the new values of the news. Thus, it is possible to update values for a specified news by calling this function with the last known *change_id*.

The MqlCalendarValue structure provides methods for checking and setting values from the actual_value, forecast_value, prev_value and revised_prev_value fields. If no value is specified, the field stores LONG_MIN (-9223372036854775808).

Please note that the values stored in these field are multiplied by one million. It means that when you receive values in MqlCalendarValue using functions CalendarValueById, CalendarValueHistoryByEvent, CalendarValueHistory, CalendarValueLastByEvent and CalendarValueLast, you should check if the field values are equal to LONG_MIN; if a value is specified in a field, then you should divide the value by 1,000,000 in order to get the value. Another method to get the values is to check and to get values using the functions of the MqlCalendarValue structure.

The sample EA listening for the economic calendar events:

```
#property description "Example of using the CalendarValueLast function"
#property description " to develop the economic calendar events listener."
#property description "To achieve this, get the current change ID"
#property description " of the Calendar database. Then, use this ID to receive"
#property description " only new events via the timer survey"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create timer
   EventSetTimer(60);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- destroy timer
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//--- Calendar database change ID
   static ulong calendar_change_id=0;
//--- first launch attribute
   static bool first=true;
//--- event value array
   MqlCalendarValue values[];
//--- perform initialization - get the current calendar_change_id
   if(first)
     {
      //--- get the Calendar database change ID
      if(CalendarValueLast(calendar_change_id,values)>0)
        {
         //--- this code block cannot be executed during the first launch but let's ac
         PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                     __FUNCTION__,calendar_change_id);
         //--- set the flag and exit before the timer's next event
         first=false;
         return;
        }
      else
        {
         //--- data are not received (this is normal for the first launch), check for
         int error_code=GetLastError();
         if(error_code==0)
           {
            PrintFormat("%s: Received the Calendar database current ID: change_id=%d",
                        __FUNCTION__,calendar_change_id);
            //--- set the flag and exit before the timer's next event
            first=false;
            //--- now we have the calendar_change_id value
            return;
           }
         else
           {
            //--- and this is really an error
            PrintFormat("%s: Failed to get events in CalendarValueLast. Error code: %
                        __FUNCTION__,error_code);
            //--- operation completed in a failure, re-initialize during the next cal
            return;
           }
        }
     }

//--- we have the last known value of the Calendar change ID (change_id)
   ulong old_change_id=calendar_change_id;
//--- check if there are new Calendar events
   if(CalendarValueLast(calendar_change_id,values)>0)
     {
      PrintFormat("%s: Received new Calendar events: %d",
                  __FUNCTION__,ArraySize(values));
      //--- display data from the 'values' array in the Journal
      ArrayPrint(values);
      //--- display the values of the previous and new Calendar IDs in the Journal
      PrintFormat("%s: Previous change_id=%d, new change_id=%d",
                  __FUNCTION__,old_change_id,calendar_change_id);
      //--- display new events in the Journal
      ArrayPrint(values);
      /*
     write your code that is to handle occurrence of events here
      */
     }
//---
  }
/*
  Example of the listener operation:
  OnTimer: Received the Calendar database current ID: change_id=33281792
  OnTimer: Received new events for the Calendar: 1
        [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 91040   76020013 2019.03.20 15:30:00 1970.01.01 00:00:00          0       -507
  OnTimer: Previous change_id=33281792, new change_id=33282048
        [id] [event_id]              [time]            [period] [revision] [actual_va
   [0] 91040   76020013 2019.03.20 15:30:00 1970.01.01 00:00:00          0       -507
  OnTimer: Received new events for the Calendar: 1
        [id] [event_id]              [time]            [period] [revision]       [actu
   [0] 91041   76020013 2019.03.27 15:30:00 1970.01.01 00:00:00          0 -922337203
  OnTimer: Previous change_id=33282048, new change_id=33282560
        [id] [event_id]              [time]            [period] [revision]       [actu
   [0] 91041   76020013 2019.03.27 15:30:00 1970.01.01 00:00:00          0 -922337203

*/
```

See also

CalendarValueLast, CalendarValueHistory, CalendarValueHistoryByEvent, CalendarValueById

Access to Timeseries and Indicator Data

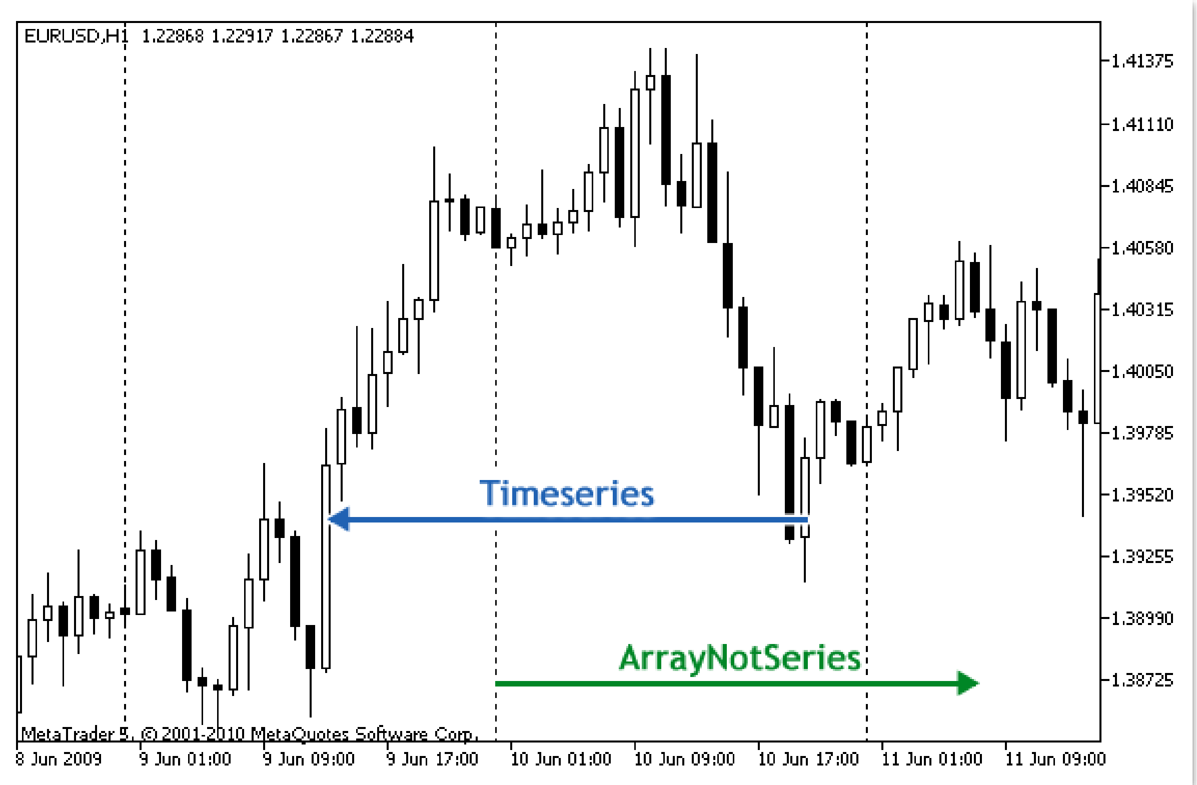
These are functions for working with timeseries and indicators. A timeseries differs from the usual data array by its reverse ordering - elements of timeseries are indexed from the end of an array to its begin (from the most recent data to the oldest ones). To copy the time-series values and indicator data, it's recommended to use dynamic arrays only, because copying functions are designed to allocate the necessary size of arrays that receive values.

There is an **important exception** to this rule: if timeseries and indicator values need to be copied often, for example at each call of OnTick() in Expert Advisors or at each call of OnCalculate() in indicators, in this case one should better use statically distributed arrays, because **operations of memory allocation** for dynamic arrays **require additional time**, and this will have effect during testing and optimization.

When using functions accessing timeseries and indicator values, indexing direction should be taken into account. This is described in the Indexing Direction in Arrays, Buffers and Timeseries section.

Access to indicator and timeseries data is implemented irrespective of the fact whether the requested data are ready (the so called asynchronous access). This is critically important for the calculation of custom indicator, so if there are no data, functions of *Copy...()* type immediately return an error. However, when accessing form Expert Advisors and scripts, several attempts to receive data are made in a small pause, which is aimed at providing some time necessary to download required timeseries or to calculate indicator values.

The Organizing Data Access section describes details of receiving, storing and requesting price data in the MetaTrader 5 client terminal.



It is historically accepted that an access to the price data in an array is performed from the end of the data. Physically, the new data are always written at the array end, but the index of the array is always equal to zero. The 0 index in the timeseries array denotes data of the current bar, i.e. the bar that corresponds to the unfinished time interval in this timeframe.

A timeframe is the time period, during which a single price bar is formed. There are 21 predefined standard timeframes.

Function	Action
SeriesInfoInteger	Returns information about the state of historical data
Bars	Returns the number of bars count in the history for a specified symbol and period
BarsCalculated	Returns the number of calculated data in an indicator buffer or -1 in the case of error (data hasn't been calculated yet)
IndicatorCreate	Returns the handle to the specified technical indicator created by an array of MqlParam type parameters
IndicatorParameters	Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters
IndicatorRelease	Removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else
CopyBuffer	Gets data of a specified buffer from a specified indicator into an array
CopyRates	Gets history data of the Rates structure for a specified symbol and period into an array
CopySeries	Gets the synchronized timeseries from the Rates structure for the specified symbol-period and the specified amount
CopyTime	Gets history data on bar opening time for a specified symbol and period into an array
CopyOpen	Gets history data on bar opening price for a specified symbol and period into an array
CopyHigh	Gets history data on maximal bar price for a specified symbol and period into an array
CopyLow	Gets history data on minimal bar price for a specified symbol and period into an array
CopyClose	Gets history data on bar closing price for a specified symbol and period into an array
CopyTickVolume	Gets history data on tick volumes for a specified symbol and period into an array
CopyRealVolume	Gets history data on trade volumes for a specified symbol and period into an array
CopySpread	Gets history data on spreads for a specified symbol and period into an array

Function	Action
CopyTicks	Gets ticks in the MqlTick format into ticks_array
CopyTicksRange	Gets ticks in the MqlTick format within the specified date range to ticks_array
iBars	Returns the number of bars of a corresponding symbol and period, available in history
iBarShift	Returns the index of the bar corresponding to the specified time
iClose	Returns the Close price of the bar (indicated by the 'shift' parameter) on the corresponding chart
iHigh	Returns the High price of the bar (indicated by the 'shift' parameter) on the corresponding chart
iHighest	Returns the index of the highest value found on the corresponding chart (shift relative to the current bar)
iLow	Returns the Low price of the bar (indicated by the 'shift' parameter) on the corresponding chart
iLowest	Returns the index of the smallest value found on the corresponding chart (shift relative to the current bar)
iOpen	Returns the Open price of the bar (indicated by the 'shift' parameter) on the corresponding chart
iTime	Returns the opening time of the bar (indicated by the 'shift' parameter) on the corresponding chart
iTickVolume	Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart
iRealVolume	Returns the real volume of the bar (indicated by the 'shift' parameter) on the corresponding chart
iVolume	Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart
iSpread	Returns the spread value of the bar (indicated by the 'shift' parameter) on the corresponding chart

Despite the fact that by using the ArraySetAsSeries() function it is possible to set up in arrays access to elements like that in timeseries, it should be remembered that the array elements are physically stored in one and the same order - only indexing direction changes. To demonstrate this fact let's perform an example:

```
   datetime TimeAsSeries[];
//--- set access to the array like to a timeseries
   ArraySetAsSeries(TimeAsSeries,true);
   ResetLastError();
   int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
   if(copied<=0)
```

```
     {
      Print("The copy operation of the open time values for last 10 bars has failed");
      return;
     }
   Print("TimeCurrent =",TimeCurrent());
   Print("ArraySize(Time) =",ArraySize(TimeAsSeries));
   int size=ArraySize(TimeAsSeries);
   for(int i=0;i<size;i++)
     {
      Print("TimeAsSeries["+i+"] =",TimeAsSeries[i]);
     }

   datetime ArrayNotSeries[];
   ArraySetAsSeries(ArrayNotSeries,false);
   ResetLastError();
   copied=CopyTime(NULL,0,0,10,ArrayNotSeries);
   if(copied<=0)
     {
      Print("The copy operation of the open time values for last 10 bars has failed");
      return;
     }
   size=ArraySize(ArrayNotSeries);
   for(int i=size-1;i>=0;i--)
     {
      Print("ArrayNotSeries["+i+"] =",ArrayNotSeries[i]);
     }
```

As a result we will get the output like this:

```
TimeCurrent = 2009.06.11 14:16:23
ArraySize(Time) = 10
TimeAsSeries[0] = 2009.06.11 14:00:00
TimeAsSeries[1] = 2009.06.11 13:00:00
TimeAsSeries[2] = 2009.06.11 12:00:00
TimeAsSeries[3] = 2009.06.11 11:00:00
TimeAsSeries[4] = 2009.06.11 10:00:00
TimeAsSeries[5] = 2009.06.11 09:00:00
TimeAsSeries[6] = 2009.06.11 08:00:00
TimeAsSeries[7] = 2009.06.11 07:00:00
TimeAsSeries[8] = 2009.06.11 06:00:00
TimeAsSeries[9] = 2009.06.11 05:00:00

ArrayNotSeries[9] = 2009.06.11 14:00:00
ArrayNotSeries[8] = 2009.06.11 13:00:00
ArrayNotSeries[7] = 2009.06.11 12:00:00
ArrayNotSeries[6] = 2009.06.11 11:00:00
ArrayNotSeries[5] = 2009.06.11 10:00:00
ArrayNotSeries[4] = 2009.06.11 09:00:00
ArrayNotSeries[3] = 2009.06.11 08:00:00
```

```
ArrayNotSeries[2] = 2009.06.11 07:00:00
ArrayNotSeries[1] = 2009.06.11 06:00:00
ArrayNotSeries[0] = 2009.06.11 05:00:00
```

As we see from the output, as the index of TimeAsSeries array increases, the time value of the index decreases, i.e. we move from the present to the past. For the common array ArrayNotSeries the result is different - as index grows, we move from past to present.

See Also

ArrayIsDynamic, ArrayGetAsSeries, ArraySetAsSeries, ArrayIsSeries

Indexing Direction in Arrays, Buffers and Timeseries

The default indexing of all arrays and indicator buffers is left to right. The index of the first element is always equal to zero. Thus, the very first element of an array or indicator buffer with index 0 is by default on the extreme left position, while the last element is on the extreme right position.

An indicator buffer is a dynamic array of type double, whose size is managed by the client terminals, so that it always corresponds to the number of bars the indicator is calculated on. A usual dynamic array of type double is assigned as an indicator buffer using the SetIndexBuffer() function. Indicator buffers do not require setting of their size using function ArrayResize() - this will be done by the executing system of the terminal.

Timeseries are arrays with reverse indexing, i.e. the first element of a timeseries is in the extreme right position, and the last element is in the extreme left position. Timeseries being used for storing history price data and contain the time information, we can say that the newest data are placed in the extreme right position of the timeseries, while the oldest data are in the extreme left position.

So the timeseries element with index 0 contains the information about the latest quote of a symbol. If a timeseries contains data on a daily timeframe, data of the current yet uncompleted day are located on the zero position, and the position with index 1 contains yesterday data.

Changing the Indexing Direction

Function ArraySetAsSeries() allows changing the method of accessing elements of a dynamic array; the physical order of data storing in the computer memory is not changed at that. This function simply changes the method of addressing array elements, so when copying one array to another using function ArrayCopy(), the contents of the recipient array will not depend on the indexing direction in the source array.

Direction of indexing cannot be changed for statically distributed arrays. Even if an array was passed as a parameter to a function, attempts to change the indexing direction inside this function will bring no effect.

For indicator buffers, like for usual arrays, indexing direction can also be set as backward (like in timeseries), i.e. reference to the zero position in the indicator buffer will mean reference to the last value on the corresponding indicator buffer and this will correspond to the value of the indicator on the latest bar. Still, the physical location of indicator bars will be unchanged.

Receiving Price Data in Indicators

Each custom indicator must necessarily contain the OnCalculate() function, to which price data required for calculating values in indicator buffers are passed. Indexing direction in these passed arrays can be found out using function ArrayGetAsSeries().

Arrays passed to the function reflect price data, i.e. these arrays have the sign of a timeseries and function ArrayIsSeries() will return true when checking these arrays. However, in any case indexing direction should be checked only by function ArrayGetAsSeries().

In order not to be dependent on default values, ArraySetAsSeries() should be unconditionally called for the arrays you are going to work with, and set the required direction.

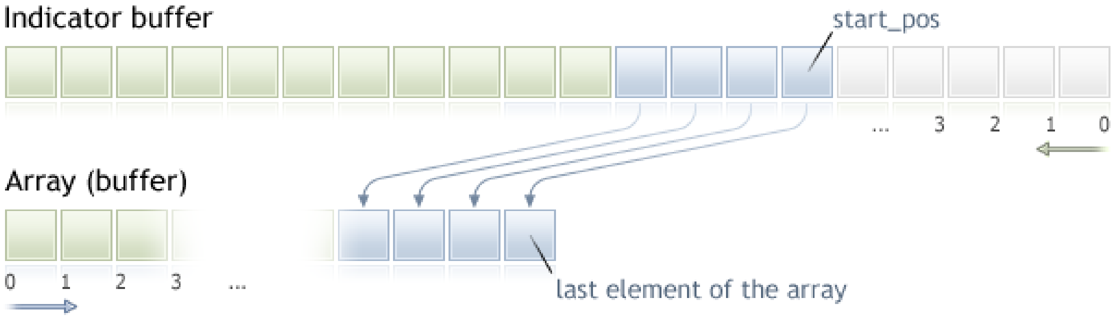
Receiving Price Data and Indicator Values

Default indexing direction of all arrays in Expert Advisors, indicators and scripts is left-to-right. If necessary, in any mql5 program you can request timeseries values on any symbol and timeframe, as well as values of indicators calculated on any symbol and timeframe.

Use functions Copy...() for these purposes:

- CopyBuffer - copy values of an indicator buffer to an array of double type;
- CopyRates - copy price history to an array of structures MqlRates;
- CopyTime - copy Time values to an array of datetime type;
- CopyOpen - copy Open values to an array of double type;
- CopyHigh - copy High values to an array of double type;
- CopyLow - copy Low values to an array of double type;
- CopyClose - copy Close values to an array of double type;
- CopyTickVolume - copy tick volumes to an array of long type;
- CopyRealVolume - copy equity volumes to a long type array;
- CopySpread - copy the spread history to an array of int type;

All these functions work in a similar way. Let's consider the data obtaining mechanism on the example of CopyBuffer(). It is implied that the indexing direction of requested data is that of timeseries, and the position with index 0 (zero) stores data of the current yet uncompleted bar. In order to get access to these data we need to copy the necessary volume of data into the recipient array, e.g. into array *buffer*.



When copying we need to specify the starting position in the source array, starting from which data will be copied to the recipient array. In case of success, the specified number of elements will be copied to the recipient array from the source array (from the indicator buffer in this case). Irrespective of the indexing value set in the recipient array, copying is always performed as is shown in the above figure.

If it is expected that price data will be handled in a loop with a large number of iterations, it is advisable that you check the fact of forced program termination using the IsStopped() function:

```
int copied=CopyBuffer(ma_handle,// Indicator handle
                      0,        // The index of the indicator buffer
                      0,        // Start position for copying
                      number,   // Number of values to copy
                      Buffer    // The array that receives the values
                      );
if(copied<0) return;
int k=0;
while(k<copied && !IsStopped())
  {
   //--- Get the value for the k index
   double value=Buffer[k];
   // ...
   // work with value
   k++;
  }
```

Example:

```
input int per=10; // period of the exponent
int ma_handle;    // indicator handle
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   ma_handle=iMA(_Symbol,0,per,0,MODE_EMA,PRICE_CLOSE);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   double ema[10];
   int copied=CopyBuffer(ma_handle,// indicator handle
                         0,        // index of the indicator buffer
                         0,        // starting position to copy from
                         10,       // number of values for copying
                         ema       // value receiving array
                         );
   if(copied<0) return;
// .... further code
  }
```

See also

[Organizing Data Access]()

Organizing Data Access

In this section questions connected with obtaining, storing and requesting price data (timeseries) are considered.

Receiving Data from a Trade Server

Before price data become available in the MetaTrader 5 terminal, they must be received and processed. To receive data, connection to the MetaTrader 5 trade server must be established. Data are received in the form of packed blocks of minute bars from the server upon the request of a terminal.

The mechanism of server reference for data doesn't depend on how the request has been initiated - by a user when navigating in a chart or in a program way in the MQL5 language.

Storing Intermediate Data

Data received from a server are automatically unpacked and saved in the HCC intermediate format. Data on each symbol are written into a separate folder: *terminal_directory*\bases*server_name*\history*symbol_name*. For example, data on EURUSD received from the MetaQuotes-Demo server will be stored in *terminal_directory*\bases\MetaQuotes-Demo\history\EURUSD\.

Data are written into files with .hcc extension. Each file stores data of minute bars for one year. For example, the file named 2009.hcc in the EURUSD folder contains minute bars of EURUSD for year 2009. These files are used for preparing price data for all timeframes and are not intended for direct access.

Obtaining Data on a Necessary Timeframe out of Intermediate Data

Intermediate HCC files are used as the data source for building price data for requested timeframes in the HC format. Data of HC format are timeseries that are maximally prepared for a quick access. They are created upon a request of a chart or a MQL5 program. The volume of data should not exceed the value of the "Max bars in charts" parameter. Data are stored for further using in files with hc extension.

To save resources, data on a timeframe are stored and saved in RAM only if necessary. If not called for a long time, they are released from RAM and saved into a file. For each timeframe, data are prepared regardless of whether there are ready data for other timeframes or not. Rules of forming and accessing data are the same for all timeframes. I.e., despite the fact that the unit data stored in HCC is one minute (M1), the availability of HCC data doesn't mean the availability of data on M1 timeframe as HC in the same volume.

Receipt of new data from a server calls automatic update of used price data in HC format of all timeframes. It also leads to the recalculation of all indicators that implicitly use them as input data for calculations.

Parameter "Max bars in chart"

The "Max bars in charts" parameter restricts number of bars in HC format available to charts, indicators and mql5 programs. This is valid for all available timeframes and serves, first of all, to save computer resources.

When setting a large value of this parameter, it should be remembered, that if deep history price data for small timeframes are available, memory used for storing timeseries and indicator buffers can become hundreds of megabytes and reach the RAM restriction for the client terminal program (2Gb for 32-bit applications of MS Windows).

Change of the "Max bars in charts" comes into effect after the client terminal is restarted. Change of this parameter causes neither automatic referring to a server for additional data, nor forming of additional bars of timeseries. Additional price data are requested from the server, and timeseries are updated taking into account the new limitation, in case of either chart scroll to the area with no data, or when data are requested by a mql5 program.

Volume of data requested from the server corresponds to the required number of bars of this timeframe with the "Max bars in charts" parameter taken into account. The restriction set by this parameter is not strict, and in some cases the number of available bars for a timeframe can be a little more than the current parameter value.

Data Availability

Presence of data on HCC format or even in the prepared for using HC format does not always denote the absolute availability of these data to be shown in a chart or to be used in MQL5 programs.

When accessing to price data or indicator values from a mql5 program it should be remembered that their availability in a certain moment of time or starting from a certain moment of time is not guaranteed. It is connected with the fact that with the purpose of saving resources, the full copy of data necessary for a mql5 program isn't stored in MetaTrader 5; only direct access to the terminal database is given.

The price history for all timeframes is built from common data of HCC format, and any update of data from a server leads to the update of data for all timeframes and to the recalculation of indicators. Due to this access to data can be closed, even if these data were available a moment ago.

Synchronization of the Terminal Data and Server Data

Since a mql5 program can call data from any symbol and timeframe, there is a possibility that data of a necessary timeseries are not formed yet in the terminal or the necessary price data aren't synchronized with the trade server. In this case it's hard to predict the latency time.

Algorithms using "do-nothing" loops are not the best solution. The only exception in this case are scripts, because they do not have any alternative algorithm choice due to not having event handling. For custom indicators such algorithms, as well as any other "do-nothing" loops are strongly not recommended, because they lead to termination of calculation of all indicators and any other handling of price data of the symbol.

For Expert Advisors and indicators, it is better to use the event model of handling. If during handling of OnTick() or OnCalculate() event, data receipt for the required timeseries failed, you should exit the event handler, relying on the access availability during the next call of the handler.

Example of a Script for Adding History

Let's consider the example of a script that executes a request to receive history for the selected symbol from a trade server. The script is intended for running in a chart of a selected symbol;

timeframe doesn't matter, because, as it was mentioned above, price data are received from a trade server as packed one minute data, from which any predefined timeseries is constructed then.

Write all actions concerning data receipt as a separate function CheckLoadHistory(symbol, timeframe, start_date):

```
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
  }
```

The CheckLoadHistory() function is designed as a universal function that can be called from any program (Expert Advisor, script or indicator); and therefore it requires three input parameters: symbol name, period and start date to indicate the beginning of price history you need.

Insert necessary checks into the function code before requesting the missing history. First of all, we should make sure that the symbol name and period value are correct:

```
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
```

Then let's make sure that the symbol is available in the MarketWatch window, i.e., the history for the symbol will be available when sending a request to a trade server. If there is no such a symbol in MarketWatch, add it using the SymbolSelect() function.

```
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {f
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
```

Now we should receive the start date of the available history for the indicated symbol/period pair. Perhaps, the value of the input parameter startdate, passed to CheckLoadHistory(), is within the available history; then request to a trade server is not needed. In order to obtain the very first date for the symbol-period as of the moment, the SeriesInfoInteger() function with the SERIES_FIRSTDATE modifier is used.

```
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
```

The next important check is checking the type of the program, from which the function is called. Note that it is not desirable to send a request to update the timeseries from indicator with the same period. The undesirability of requesting data on the same symbol-period as that of the indicator is conditioned by the fact that update of history data is performed in the same thread where the indicator operates. So the possibility of deadlock occurrence is high. To check this use the MQL5InfoInteger() function with the MQL5_PROGRAM_TYPE modifier.

```
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
      return(-4);
```

If all the checks have been passed successfully, make the last attempt to do without referring to the trade server. First let's find out the start date, for which minute data in HCC format are available. Request this value using the SeriesInfoInteger() function with the SERIES_TERMINAL_FIRSTDATE modifier and again compare it to the value of the start_date parameter.

```
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- there is loaded data to build timeseries
      if(first_date>0)
        {
         //--- force timeseries build
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- check date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
```

If after all the checks the execution thread is still in the body of the CheckLoadHistory() function, it means there is a necessity to request the missing price data from a trade server. First, return the value of "Max bars in chart" using the TerminalInfoInteger() function:

```
  int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
```

We'll need it to prevent requesting extra data. Then find the very first date in the symbol history on the trade server (regardless of the period) using already known function SeriesInfoInteger() with the SERIES_SERVER_FIRSTDATE modifier.

```
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_dat
      Sleep(5);
```

Since the request is an asynchronous operation, the function is called in the loop with a small delay of 5 milliseconds until the first_server_date variable receives a value, or the loop execution is terminated by a user (IsStopped() will return *true* in this case). Let's indicate a correct value of the start date, starting from which we request price data from a trade server.

```
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date," for ",
symbol," does not match to first series date ",first_date);
```

If the start date *first_server_date* of the server is lower than the start date *first_date* of the symbol in HCC format, the corresponding entry will be output in the journal.

Now we are ready to make a request to a trade server asking for missing price data. Make the request in the form of a loop and start filling out its body:

```
   while(!IsStopped())
     {
      //1. wait for synchronization between the re-built timeseries and intermediate
      //2. receive the current number of bars in this timeseries
      //    if bars is larger than Max_bars_in_chart, we can exit, work is over
      //3. obtain the start date first_date in the re-built timeseries and compare it
      //    if first_date is lower than start_date, we can exit, work is over
      //4. request from a server a new part of history - 100 bars starting from last
     }
```

The first three points are implemented by already known means.

```
   while(!IsStopped())
     {
      //--- 1.wait till timeseries re-build process is over
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- 2.request how many bars we have
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         //--- bars more than ones that can be drawn in the chart, exit
         if(bars>=max_bars) return(-2);
         //--- 3. return the current start date in the timeseries
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            // start date was earlier than that requested, task completed
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //4. Request from a server a new part of history - 100 bars starting from last
     }
```

The last fourth point is left - requesting history. We can't refer to a server directly, but any Copy-function automatically initiates request sending to a server, if the history in HCC format is not enough. Since the time of the very first start date in the *first_date* variable is the simple and natural criterion to evaluate the request execution degree, then the easiest way is to use the CopyTime() function.

When calling functions that copy any data from timeseries, it should be noted that the *start* parameter (number of the bar, starting from which price data should be copied) must always be within the available terminal history. If you have only 100 bars, it meaningless to try copying 300 bars starting from the bar with the index 500. Such a request will be understood as an erroneous and won't be handled, i.e. no additional history will be loaded from a trade server.

That's why we'll copy bars in groups of 100 starting from the bar with the *bars* index. This will provide the smooth loading of missing history from a trade server. Actually a little more than the requested 100 bars will be loaded, while server sends oversized history.

```
   int copied=CopyTime(symbol,period,bars,100,times);
```

After the copying operation, we should analyze the number of copied elements. If the attempt fails, then value of the *copied* will be equal to null and the value of the fail_cnt counter will be increased by 1. After 100 failing attempts, the operation of the function will be stopped.

```
int fail_cnt=0;
...
   int copied=CopyTime(symbol,period,bars,100,times);
   if(copied>0)
     {
      //--- check data
      if(times[0]<=start_date)  return(0);  // the copied value is smaller, ready
      if(bars+copied>=max_bars) return(-2); // bars are more than can be drawn in the
      fail_cnt=0;
     }
   else
     {
      //--- no more than 100 failing attempts in succession
      fail_cnt++;
      if(fail_cnt>=100) return(-5);
      Sleep(10);
     }
```

So, not only correct handling of the current situation at each moment of execution is implemented in the function, but also the termination code is returned, that can be handled after calling the CheckLoadHistory() function for getting additional information. For example, this way:

```
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Unknown symbol ",InpLoadedSymbol);                      break;
      case -2 : Print("More requested bars than can be drawn in the chart"); break;
      case -3 : Print("Execution stopped by user");                          break;
      case -4 : Print("Indicator mustn't load its own data");                break;
      case -5 : Print("Loading failed");                                     break;
      case  0 : Print("All data loaded");                                    break;
      case  1 : Print("Already available data in timeseries are enough");    break;
      case  2 : Print("Timeseries is built from available terminal data");   break;
      default : Print("Execution result undefined");
     }
```

The full code of the function can be found in the example of a script that shows the correct organization of access to any data with the handling of request's results.

Code:

```
//+------------------------------------------------------------------+
//|                                              TestLoadHistory.mq5 |
//|                         Copyright 2009, MetaQuotes Software Corp. |
//|                                               https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.02"
#property script_show_inputs
//--- input parameters
input string          InpLoadedSymbol="NZDUSD";   // Symbol to be load
input ENUM_TIMEFRAMES InpLoadedPeriod=PERIOD_H1;  // Period to be loaded
input datetime        InpStartDate=D'2006.01.01'; // Start date
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   Print("Start load",InpLoadedSymbol+","+GetPeriodName(InpLoadedPeriod),"from",InpSta
//---
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Unknown symbol ",InpLoadedSymbol);             break;
      case -2 : Print("Requested bars more than max bars in chart"); break;
      case -3 : Print("Program was stopped");                        break;
      case -4 : Print("Indicator shouldn't load its own data");      break;
      case -5 : Print("Load failed");                                break;
      case  0 : Print("Loaded OK");                                  break;
      case  1 : Print("Loaded previously");                          break;
      case  2 : Print("Loaded previously and built");                break;
      default : Print("Unknown result");
     }
//---
   datetime first_date;
   SeriesInfoInteger(InpLoadedSymbol,InpLoadedPeriod,SERIES_FIRSTDATE,first_date);
   int bars=Bars(InpLoadedSymbol,InpLoadedPeriod);
   Print("First date ",first_date," - ",bars," bars");
//---
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
   datetime first_date=0;
   datetime times[100];
//--- check symbol & period
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
//--- check if symbol is selected in the Market Watch
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
//--- check if data is present
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
//--- don't ask for load of its own data if it is an indicator
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
      return(-4);
//--- second attempt
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- there is loaded data to build timeseries
      if(first_date>0)
        {
         //--- force timeseries build
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- check date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
//--- max bars in chart from terminal options
   int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
//--- load symbol history info
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date
      Sleep(5);
//--- fix start date for loading
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date," for ",symbol,
            " does not match to first series date ",first_date);
//--- load data step by step
   int fail_cnt=0;
   while(!IsStopped())
     {
      //--- wait for timeseries build
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- ask for built bars
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         if(bars>=max_bars) return(-2);
         //--- ask for first date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //--- copying of next part forces data loading
      int copied=CopyTime(symbol,period,bars,100,times);
      if(copied>0)
        {
         //--- check for data
         if(times[0]<=start_date)  return(0);
         if(bars+copied>=max_bars) return(-2);
         fail_cnt=0;
        }
      else
        {
         //--- no more than 100 failed attempts
         fail_cnt++;
```

```
          if(fail_cnt>=100) return(-5);
          Sleep(10);
         }
      }
//--- stopped
   return(-3);
  }
//+------------------------------------------------------------------+
//| Returns string value of the period                               |
//+------------------------------------------------------------------+
string GetPeriodName(ENUM_TIMEFRAMES period)
  {
   if(period==PERIOD_CURRENT) period=Period();
//---
   switch(period)
     {
      case PERIOD_M1:  return("M1");
      case PERIOD_M2:  return("M2");
      case PERIOD_M3:  return("M3");
      case PERIOD_M4:  return("M4");
      case PERIOD_M5:  return("M5");
      case PERIOD_M6:  return("M6");
      case PERIOD_M10: return("M10");
      case PERIOD_M12: return("M12");
      case PERIOD_M15: return("M15");
      case PERIOD_M20: return("M20");
      case PERIOD_M30: return("M30");
      case PERIOD_H1:  return("H1");
      case PERIOD_H2:  return("H2");
      case PERIOD_H3:  return("H3");
      case PERIOD_H4:  return("H4");
      case PERIOD_H6:  return("H6");
      case PERIOD_H8:  return("H8");
      case PERIOD_H12: return("H12");
      case PERIOD_D1:  return("Daily");
      case PERIOD_W1:  return("Weekly");
      case PERIOD_MN1: return("Monthly");
     }
//---
   return("unknown period");
  }
```

SeriesInfoInteger

Returns information about the state of historical data. There are 2 variants of function calls.

Directly returns the property value.

```
long  SeriesInfoInteger(
   string                     symbol_name,     // symbol name
   ENUM_TIMEFRAMES            timeframe,       // period
   ENUM_SERIES_INFO_INTEGER   prop_id,         // property identifier
   );
```

Returns true or false depending on the success of the function run.

```
bool  SeriesInfoInteger(
   string                     symbol_name,     // symbol name
   ENUM_TIMEFRAMES            timeframe,       // period
   ENUM_SERIES_INFO_INTEGER   prop_id,         // property ID
   long&                      long_var         // variable for getting info
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

prop_id

[in] Identifier of the requested property, value of the ENUM_SERIES_INFO_INTEGER enumeration.

long_var

[out] Variable to which the value of the requested property is placed.

Return Value

In the first case, it returns value of the long type.

For the second case, it returns true, if the specified property is available and its value has been placed into long_var variable, otherwise it returns false. For more details about an error, call GetLastError().

Example:

```
void OnStart()
  {
//---
   Print("Total number of bars for the symbol-period at this moment = ",
         SeriesInfoInteger(Symbol(),Period(),SERIES_BARS_COUNT));

   Print("The first date for the symbol-period at this moment = ",
         (datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_FIRSTDATE));
   Print("The first date in the history for the symbol-period on the server = ",
         (datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_SERVER_FIRSTDATE));

   Print("Symbol data are synchronized = ",
         (bool)SeriesInfoInteger(Symbol(),Period(),SERIES_SYNCHRONIZED));
  }
```

Bars

Returns the number of bars count in the history for a specified symbol and period. There are 2 variants of functions calls.

Request all of the history bars

```
int  Bars(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe        // period
   );
```

Request the history bars for the selected time interval

```
int  Bars(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time        // end date and time
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_time

[in] Bar time corresponding to the first element.

stop_time

[in] Bar time corresponding to the last element.

Return Value

If the start_time and stop_time parameters are defined, the function returns the number of bars in the specified time interval, otherwise it returns the total number of bars.

Note

If data for the timeseries with specified parameters are not formed in the terminal by the time of the Bars() function call, or data of the timeseries are not synchronized with a trade server by the moment of the function call, the function returns a zero value.

When requesting the number of bars in a specified time interval, only bars with an open time falling within the interval are considered. For example, if the current day of the week is Saturday and the request is made for the number of W1 bars with start_time=last_tuesday and stop_time=last_friday, the function will return 0 since the open time of a W1 timeframe is always Sunday and not a single W1 bar falls within the specified interval.

Sample request for the number of all history bars:

```
int bars=Bars(_Symbol,_Period);
if(bars>0)
  {
   Print("Number of bars in the terminal history for the symbol-period at the momen
  }
else  //no available bars
  {
   //--- data on the symbol might be not synchronized with data on the server
   bool synchronized=false;
   //--- loop counter
   int attempts=0;
   // make 5 attempts to wait for synchronization
   while(attempts<5)
     {
      if(SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED))
        {
         //--- synchronization done, exit
         synchronized=true;
         break;
        }
      //--- increase the counter
      attempts++;
      //--- wait 10 milliseconds till the next iteration
      Sleep(10);
     }
   //--- exit the loop after synchronization
   if(synchronized)
     {
      Print("Number of bars in the terminal history for the symbol-period at the mo
      Print("The first date in the terminal history for the symbol-period at the mo
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));
      Print("The first date in the history for the symbol on the server = ",
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
     }
   //--- synchronization of data didn't happen
   else
     {
      Print("Failed to get number of bars for ",_Symbol);
     }
  }
```

Sample request for the number of bars in the specified interval:

```
int n;
datetime date1 = D'2016.09.02 23:55'; // Friday
datetime date2 = D'2016.09.05 00:00'; // Monday
datetime date3 = D'2016.09.08 00:00'; // Thursday
//---
n=Bars(_Symbol,PERIOD_H1,D'2016.09.02 02:05',D'2016.09.02 10:55');
```

```
   Print("Number of bars: ",n); // Output: "Number of bars: 8", H2 bar is considered 
   n=Bars(_Symbol,PERIOD_D1,date1,date2);
   Print("Number of bars: ",n); // Output: "Number of bars: 1", since an open time of
   n=Bars(_Symbol,PERIOD_W1,date2,date3);
   Print("Number of bars: ",n); // Output: "Number of bars: 0", since not a single W1
```

See also

[Event Handling Functions](#)

BarsCalculated

Returns the number of calculated data for the specified indicator.

```
int  BarsCalculated(
   int       indicator_handle,     // indicator handle
   );
```

Parameters

indicator_handle

[in] The indicator handle, returned by the corresponding indicator function.

Return Value

Returns the amount of calculated data in the indicator buffer or -1 in the case of error (data not calculated yet)

Note

The function is useful when it's necessary to get the indicator data immediately after its creation (indicator handle is available).

Example:

```
void OnStart()
  {
   double Ups[];
//--- set timeseries ordering for the arrays
   ArraySetAsSeries(Ups,true);
//--- create handle for the Fractal Indicator
   int FractalsHandle=iFractals(NULL,0);
//--- reset the error code
   ResetLastError();
//--- try to copy the indicator values
   int i,copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Sleep(50);
      for(i=0;i<100;i++)
        {
         if(BarsCalculated(FractalsHandle)>0)
            break;
         Sleep(50);
        }
      copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
      if(copied<=0)
        {
         Print("Failed to copy upper fractals. Error = ",GetLastError(),
         "i = ",i,"    copied = ",copied);
         return;
        }
        else
          Print("Upper fractals copied",
          "i = ",i,"     copied = ",copied);
      }
    else Print("Upper fractals copied. ArraySize = ",ArraySize(Ups));
  }
```

IndicatorCreate

The function returns the handle of a specified technical indicator created based on the array of parameters of MqlParam type.

```
int  IndicatorCreate(
   string             symbol,                      // symbol name
   ENUM_TIMEFRAMES    period,                      // timeframe
   ENUM_INDICATOR     indicator_type,              // indicator type from the enum
   int                parameters_cnt=0,            // number of parameters
   const MqlParam&    parameters_array[]=NULL,     // array of parameters
   );
```

Parameters

symbol

[in] Name of a symbol, on data of which the indicator is calculated. NULL means the current symbol.

period

[in] The value of the timeframe can be one of values of the ENUM_TIMEFRAMES enumeration, 0 means the current timeframe.

indicator_type

[in] Indicator type, can be one of values of the ENUM_INDICATOR enumeration.

parameters_cnt

[in] The number of parameters passed in the parameters_array[] array. The array elements have a special structure type MqlParam. By default, zero - parameters are not passed. If you specify a non-zero number of parameters, the parameter *parameters_array* is obligatory. You can pass no more than 64 parameters.

parameters_array[]=NULL

[in] An array of MqlParam type, whose elements contain the type and value of each input parameter of a technical indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE.

Note

If the indicator handle of IND_CUSTOM type is created, the *type* field of the first element of the array of input parameters *parameters_array* must have the TYPE_STRING value of the ENUM_DATATYPE enumeration, and the *string_value* field of the first element must contain the name of the custom indicator. The custom indicator must be compiled (file with EX5 extension) and located in the directory MQL5/Indicators of the client terminal or in a subdirectory.

Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a constant string. For all other cases (use of the IndicatorCreate() function or use of a non-constant string in the parameter that sets the indicator name) the property #property tester_indicator is required:

```
#property tester_indicator "indicator_name.ex5"
```

If the first form of the call is used in a custom indicator, you can additionally indicate as the last parameter on what data it will be calculated when passing input parameters. If the "Apply to" parameter is not specified explicitly, the default calculation is based on the PRICE_CLOSE values.

Example:

```
void OnStart()
  {
   MqlParam params[];
   int      h_MA,h_MACD;
//--- create iMA("EURUSD",PERIOD_M15,8,0,MODE_EMA,PRICE_CLOSE);
   ArrayResize(params,4);
//--- set ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=8;
//--- set ma_shift
   params[1].type         =TYPE_INT;
   params[1].integer_value=0;
//--- set ma_method
   params[2].type         =TYPE_INT;
   params[2].integer_value=MODE_EMA;
//--- set applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=PRICE_CLOSE;
//--- create MA
   h_MA=IndicatorCreate("EURUSD",PERIOD_M15,IND_MA,4,params);
//--- create iMACD("EURUSD",PERIOD_M15,12,26,9,h_MA);
   ArrayResize(params,4);
//--- set fast ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=12;
//--- set slow ma_period
   params[1].type         =TYPE_INT;
   params[1].integer_value=26;
//--- set smooth period for difference
   params[2].type         =TYPE_INT;
   params[2].integer_value=9;
//--- set indicator handle as applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=h_MA;
//--- create MACD based on moving average
   h_MACD=IndicatorCreate("EURUSD",PERIOD_M15,IND_MACD,4,params);
//--- use indicators
//--- . . .
//--- release indicators (first h_MACD)
   IndicatorRelease(h_MACD);
   IndicatorRelease(h_MA);
  }
```

IndicatorParameters

Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters.

```
int  IndicatorParameters(
   int                  indicator_handle,     // indicator handle
   ENUM_INDICATOR&      indicator_type,       // a variable for receiving the indicator t
   MqlParam&            parameters[]          // an array for receiving parameters
   );
```

Parameters

indicator_handle

[in] The handle of the indicator, for which you need to know the number of parameters its is calculated on.

indicator_type

[out] A variable if the ENUM_INDICATOR type, into which the indicator type will be written.

parameters[]

[out] A dynamic array for receiving values of the MqlParam type, into which the list of indicator parameters will be written. The array size is returned by the IndicatorParameters() function.

Return Value

The number of input parameters of the indicator with the specified handle. In case of an error returns -1. For more details about the error call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {

//--- The number of windows on the chart (at least one main window is always present)
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
//--- Go through the chart windows
   for(int w=0;w<windows;w++)
     {
      //--- The number of indicators in this window/subwindow
      int total=ChartIndicatorsTotal(0,w);
      //--- Take all indicators in the window
      for(int i=0;i<total;i++)
        {
         //--- Get the short name of the indicator
         string name=ChartIndicatorName(0,w,i);
         //--- Get the indicator handle
         int handle=ChartIndicatorGet(0,w,name);
         //--- Add to log
         PrintFormat("Window=%d,  indicator #%d,  handle=%d",w,i,handle);
         //---
         MqlParam parameters[];
         ENUM_INDICATOR indicator_type;
         int params=IndicatorParameters(handle,indicator_type,parameters);
         //--- The header of the message
         string par_info="Short name "+name+", type "
                         +EnumToString(ENUM_INDICATOR(indicator_type))+"\r\n";
         //---
         for(int p=0;p<params;p++)
           {
            par_info+=StringFormat("parameter %d: type=%s, long_value=%d, double_value
                                   p,
                                   EnumToString((ENUM_DATATYPE)parameters[p].type),
                                   parameters[p].integer_value,
                                   parameters[p].double_value,
                                   parameters[p].string_value
                                   );
           }
         Print(par_info);
        }
      //--- Done for all indicators in the window
     }
//---
  }
```

See also

ChartIndicatorGet()

IndicatorRelease

The function removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else.

```
bool  IndicatorRelease(
   int       indicator_handle     // indicator handle
   );
```

Return Value

Returns true in case of success, otherwise returns false.

Note

The function allows removing an indicator handle, if it's no longer needed, thus saving memory. The handle is removed immediately, the calculation block is deleted in some time (if it's not called anymore).

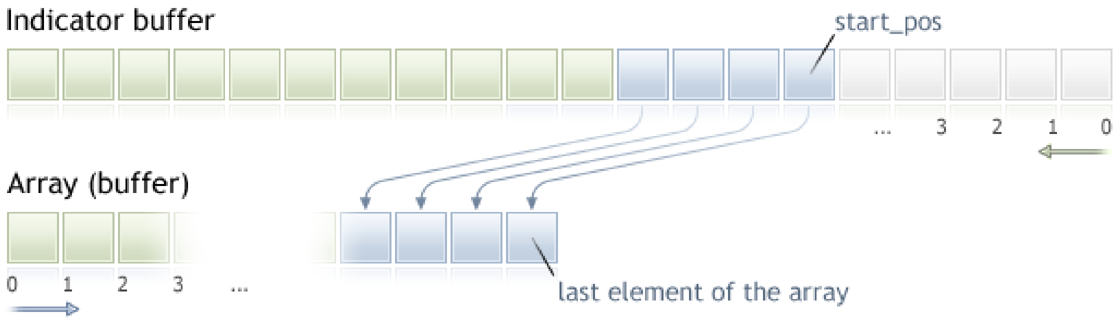
When working in the strategy tester, the IndicatorRelease() function is not executed.

Example:

```
//+------------------------------------------------------------------+
//|                                       Test_IndicatorRelease.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input int                MA_Period=15;
input int                MA_shift=0;
input ENUM_MA_METHOD     MA_smooth=MODE_SMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
//--- will store indicator handle
int MA_handle=INVALID_HANDLE;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create indicator handle
   MA_handle=iMA(Symbol(),0,MA_Period,MA_shift,MA_smooth,PRICE_CLOSE);
//--- delete global variable
   if(GlobalVariableCheck("MA_value"))
      GlobalVariableDel("MA_value");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- if the global variable value does not exist
   if(!GlobalVariableCheck("MA_value"))
     {
      //--- obtain the indicator value in the last two bars
      if(MA_handle!=INVALID_HANDLE)
        {
         //--- dynamic array for the indicator values
         double values[];
         if(CopyBuffer(MA_handle,0,0,2,values)==2 && values[0]!=EMPTY_VALUE)
           {
            //--- remember in the global variable value on the last but one bar
            if(GlobalVariableSet("MA_value",values[0]))
              {
               //--- free the handle of the indicator
               if(!IndicatorRelease(MA_handle))
                  Print("IndicatorRelease() failed. Error ",GetLastError());
               else MA_handle=INVALID_HANDLE;
              }
            else
               Print("GlobalVariableSet failed. Error ",GetLastError());
           }
        }
     }
//---
  }
```

CopyBuffer

Gets data of a specified buffer of a certain indicator in the necessary quantity.



Counting of elements of copied data (indicator buffer with the index buffer_num) from the starting position is performed from the present to the past, i.e., starting position of 0 means the current bar (indicator value for the current bar).

When copying the yet unknown amount of data, it is recommended to use a dynamic array as a buffer[] recipient buffer, because the CopyBuffer() function tries to allocate the size of the receiving array to the size of the copied data. If an indicator buffer (array that is pre-allocated for storing indicator values by the SetIndexBufer() function) is used as the buffer[] recipient array, partial copying is allowed. An example can be found in the Awesome_Oscillator.mql5 custom indicator in the standard terminal package.

If you need to make a partial copy of the indicator values into another array (non-indicator buffer), you should use an intermediate array, to which the desired number is copied. After that conduct the element-wise copying of the required number of values into the required places of a receiving array from this intermediate one.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyBuffer(
   int       indicator_handle,     // indicator handle
   int       buffer_num,           // indicator buffer number
   int       start_pos,            // start position
   int       count,                // amount to copy
   double    buffer[]              // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyBuffer(
   int       indicator_handle,     // indicator handle
   int        buffer_num,           // indicator buffer number
   datetime   start_time,           // start date and time
   int        count,                // amount to copy
   double     buffer[]              // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyBuffer(
   int        indicator_handle,     // indicator handle
   int        buffer_num,           // indicator buffer number
   datetime   start_time,           // start date and time
   datetime   stop_time,            // end date and time
   double     buffer[]              // target array to copy
   );
```

Parameters

indicator_handle

[in] The indicator handle, returned by the corresponding indicator function.

buffer_num

[in] The indicator buffer number.

start_pos

[in] The position of the first element to copy.

count

[in] Data count to copy.

start_time

[in] Bar time, corresponding to the first element.

stop_time

[in] Bar time, corresponding to the last element.

buffer[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration.

Example:

```
//+------------------------------------------------------------------+
//|                                              TestCopyBuffer3.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool               AsSeries=true;
input int                period=15;
input ENUM_MA_METHOD     smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int                shift=0;
//--- indicator buffers
double                   MABuffer[];
int                      ma_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Parameter AsSeries = ",AsSeries);
   Print("Indicator buffer after SetIndexBuffer() is a timeseries = ",
         ArrayGetAsSeries(MABuffer));
//--- set short indicator name
   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//--- set AsSeries (depends on input parameter)
   ArraySetAsSeries(MABuffer,AsSeries);
   Print("Indicator buffer after ArraySetAsSeries(MABuffer,true); is a timeseries = ",
         ArrayGetAsSeries(MABuffer));
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- check if all data calculated
   if(BarsCalculated(ma_handle)<rates_total) return(0);
//--- we can copy not all data
   int to_copy;
   if(prev_calculated>rates_total || prev_calculated<=0) to_copy=rates_total;
   else
     {
      to_copy=rates_total-prev_calculated;
      //--- last value is always copied
      to_copy++;
     }
//--- try to copy
   if(CopyBuffer(ma_handle,0,0,to_copy,MABuffer)<=0) return(0);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

The above example illustrates how an indicator buffer is filled out with the values of another indicator buffer from the indicator on the same symbol/period.

See a detailed example of history requesting data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

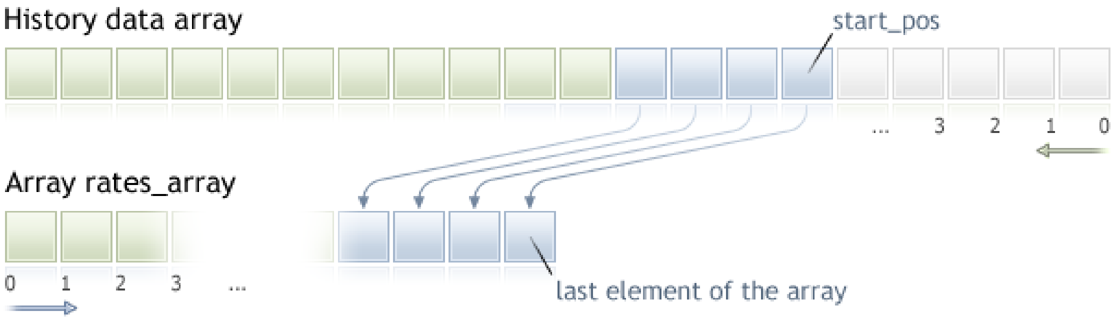
- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

See also

Properties of Custom Indicators, SetIndexBuffer

CopyRates

Gets history data of MqlRates structure of a specified symbol-period in specified quantity into the rates_array array. The elements ordering of the copied data is from present to the past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyRates(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   MqlRates         rates_array[]      // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyRates(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   MqlRates         rates_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyRates(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   datetime         stop_time,         // end date and time
   MqlRates         rates_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_time

[in] Bar time for the first element to copy.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

rates_array[]

[out] Array of MqlRates type.

Return Value

Returns the number of copied elements or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

```
void OnStart()
  {
//---
   MqlRates rates[];
   ArraySetAsSeries(rates,true);
   int copied=CopyRates(Symbol(),0,0,100,rates);
   if(copied>0)
     {
      Print("Bars copied: "+copied);
      string format="open = %G, high = %G, low = %G, close = %G, volume = %d";
      string out;
      int size=fmin(copied,10);
      for(int i=0;i<size;i++)
        {
         out=i+":"+TimeToString(rates[i].time);
         out=out+" "+StringFormat(format,
                                  rates[i].open,
                                  rates[i].high,
                                  rates[i].low,
                                  rates[i].close,
                                  rates[i].tick_volume);
         Print(out);
        }
     }
   else Print("Failed to get history data for the symbol ",Symbol());
  }
```

See a detailed example of requesting history data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,

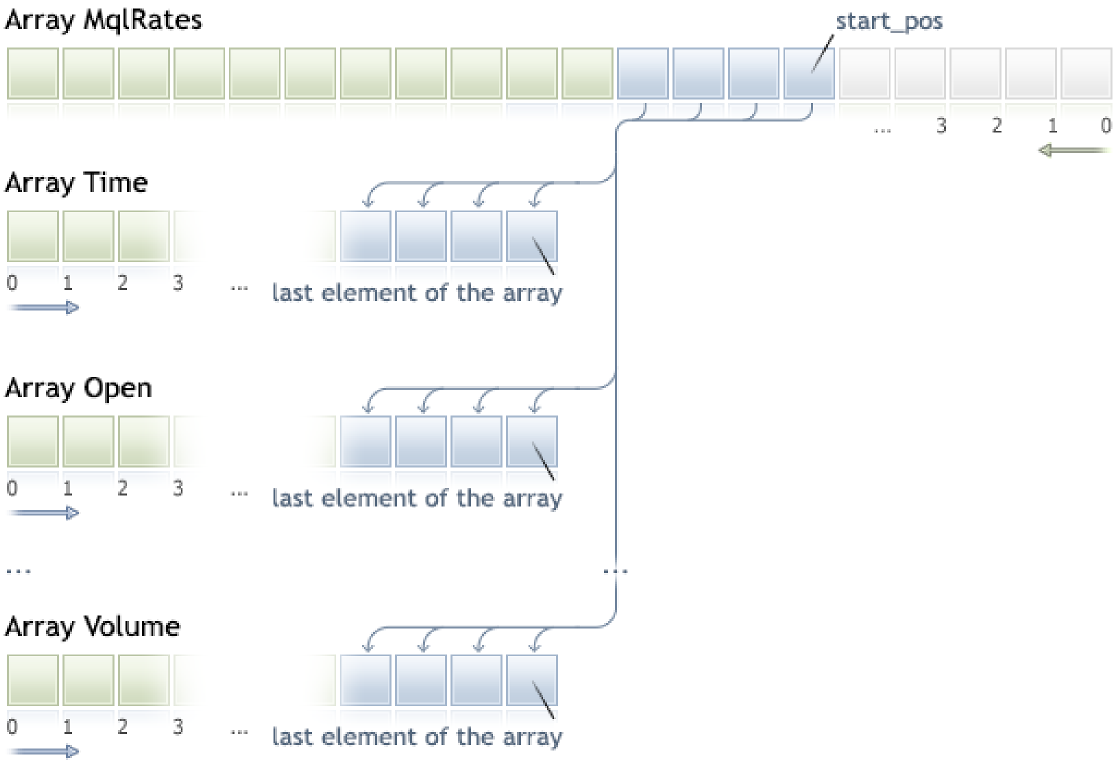
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

See also

Structures and Classes, TimeToString, StringFormat

CopySeries

Gets the synchronized timeseries from the [MqlRates](#) structure for the specified symbol-period and the specified amount. The data is received into the specified set of arrays. Elements are counted down from the present to the past, which means that the starting position equal to 0 means the current bar.


Array MqlRates
start_pos
...
3
2
1
0
Array Time
0
1
2
3
...
last element of the array
Array Open
0
1
2
3
...
last element of the array
...
...
Array Volume
0
1
2
3
...
last element of the array

If the data amount to be copied is unknown, it is recommended to use the [dynamic array](#) for the receiving arrays, since if the data amount exceeds what an array can contain, this can cause the attempt to redistribute the array to fit all of the requested data.

If you need to copy a predetermined amount of data, it is recommended to use a [statistically allocated buffer](#) to avoid unnecessary memory reallocation.

The property of the receiving array — as_series=true or as_series=false — will be ignored: during copying, the oldest timeseries element will be copied to the beginning of the physical memory allocated for the array.

```
int  CopySeries(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // amount to copy
   ulong            rates_mask,        // combination of flags to specify the requeste
   void&            array1[],          // array to receive the data of the first copie
   void&            array2[]           // array to receive the data of the second copi
   ...
   );
```

Parameters

symbol_name

[in] Symbol.

timeframe

[in] Period.

start_pos

[in] First copied element index.

count

[in] Number of copied elements.

rates_mask

[in] A combination of flags from the ENUM_COPY_RATES enumeration.

array1, array2,...

[out] Array of the appropriate type to receive the timeseries from the MqlRates structure. The order of the arrays passed to the function must match the order of the fields in the MqlRates structure.

Return Value

The number of copied elements or -1 if error occurs.

Note

If the entire interval of the requested data is beyond the data available on the server, the function returns -1. If the requested data is beyond TERMINAL_MAXBARS (the maximum number of bars on the chart), the function also returns -1.

When requesting data from an indicator, the function immediately returns -1 if requested timeseries are not constructed yet or they should be downloaded from the server. However, this will initiate data download/constructing itself.

When requesting data from an Expert Advisor or a script, download from the server is initiated if the terminal does not have the appropriate data locally, or construction of the necessary timeseries starts if the data can be constructed from the local history but they are not ready yet. The function returns the amount of data that is ready by the time the timeout expires, however the history download continues, and the function returns more data during the next similar request.

Difference between CopySeries and CopyRates

The CopySeries function allows obtaining only the necessary timeseries into different specified arrays during one call, while all of timeseries data will be synchronized. This means that all values in the resulting arrays at a certain index N will belong to the same bar on the specified Symbol/Timeframe pair. Therefore, there is no need for the programmer to ensure the synchronization of all received timeseries by the bar opening time.

Unlike CopyRates, which returns the full set of timeseries as an MqlRates array, the CopySeries function allows the programmer to get only the required timeseries as separate arrays. This can be done by specifying a combination of flags to select the type of timeseries. The order of the arrays passed to the function must match the order of the fields in the MqlRates structure:

```
struct MqlRates
  {
   datetime time;         // period start time
   double   open;         // open price
   double   high;         // high price for the period
   double   low;          // low price for the period
   double   close;        // close price
   long     tick_volume;  // tick volume
   int      spread;       // spread
   long     real_volume;  // exchange volume
  }
```

Thus, if you need to get the values of the *time*, *close* and *real_volume* timeseries for the last 100 bars of the current Symbol/Timeframe, you should use the following call:

```
datetime time[];
double   close[];
long     volume[];
CopySeries(NULL,0,0,100,COPY_RATES_TIME|COPY_RATES_CLOSE|COPY_RATES_VOLUME_REAL,time,
```

Mind the order of the arrays "*time*, *close*, *volume*" – it must match the order of the fields in the MqlRates structure. The order of values in the *rates_mask* does not matter. The mask could be as follows:

```
COPY_RATES_VOLUME_REAL|COPY_RATES_TIME|COPY_RATES_CLOSE
```

Example:

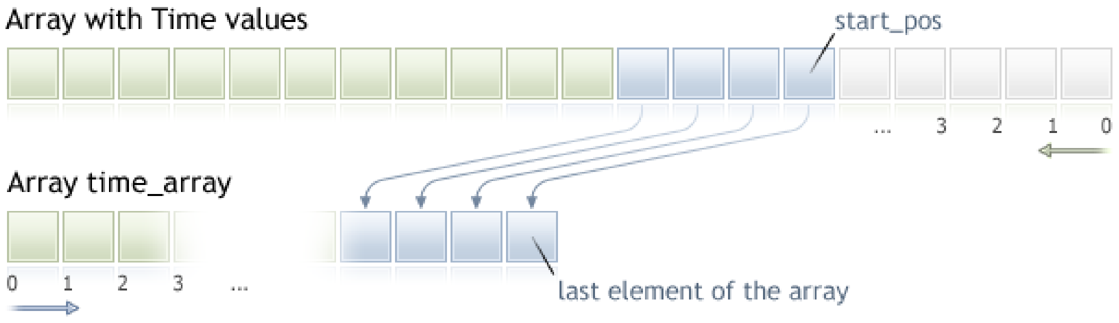
```
//--- input parameters
input datetime InpDateFrom=D'2022.01.01 00:00:00';
input datetime InpDateTo  =D'2023.01.01 00:00:00';
input uint     InpCount   =20;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- arrays to get timeseries from the MqlRates price structure
   double   open[];
   double   close[];
   float    closef[];
   datetime time1[], time2[];
//--- request close prices to a double array
   ResetLastError();
   int res1=CopySeries(NULL, PERIOD_CURRENT, 0, InpCount,
                       COPY_RATES_TIME|COPY_RATES_CLOSE, time1, close);
   PrintFormat("1. CopySeries  returns %d values. Error code=%d", res1, GetLastError()
   ArrayPrint(close);

//--- now also request open prices; use float array for close prices
   ResetLastError();
   int res2=CopySeries(NULL, PERIOD_CURRENT, 0, InpCount,
                       COPY_RATES_TIME|COPY_RATES_CLOSE|COPY_RATES_OPEN, time2, open,
   PrintFormat("2. CopySeries  returns %d values. Error code=%d", res2, GetLastError()
   ArrayPrint(closef);
//--- Compare the received data
   if((res1==res2) && (time1[0]==time2[0]))
     {
      Print("  | Time             |    Open      | Close double | Close float |");
      for(int i=0; i<10; i++)
        {
         PrintFormat("%d | %s |   %.5f    |   %.5f    |   %.5f   |",
                     i, TimeToString(time1[i]), open[i], close[i], closef[i]);
        }
     }
//--- Result
      1. CopySeries  returns 20 values. Error code=0
      [ 0] 1.06722 1.06733 1.06653 1.06520 1.06573 1.06649 1.06694 1.06675 1.06684 1.
      [10] 1.06514 1.06557 1.06456 1.06481 1.06414 1.06394 1.06364 1.06386 1.06239 1.
      2. CopySeries  returns 20 values. Error code=0
      [ 0] 1.06722 1.06733 1.06653 1.06520 1.06573 1.06649 1.06694 1.06675 1.06684 1.
      [10] 1.06514 1.06557 1.06456 1.06481 1.06414 1.06394 1.06364 1.06386 1.06239 1.
        | Time             |    Open      | Close double | Close float |
      0 | 2023.03.01 17:00 |   1.06660    |   1.06722    |   1.06722   |
      1 | 2023.03.01 18:00 |   1.06722    |   1.06733    |   1.06733   |
      2 | 2023.03.01 19:00 |   1.06734    |   1.06653    |   1.06653   |
      3 | 2023.03.01 20:00 |   1.06654    |   1.06520    |   1.06520   |
      4 | 2023.03.01 21:00 |   1.06520    |   1.06573    |   1.06573   |
      5 | 2023.03.01 22:00 |   1.06572    |   1.06649    |   1.06649   |
      6 | 2023.03.01 23:00 |   1.06649    |   1.06694    |   1.06694   |
      7 | 2023.03.02 00:00 |   1.06683    |   1.06675    |   1.06675   |
      8 | 2023.03.02 01:00 |   1.06675    |   1.06684    |   1.06684   |
      9 | 2023.03.02 02:00 |   1.06687    |   1.06604    |   1.06604   |
//---
  }
```

See also

Structures and classes, CopyRates

CopyTime

The function gets to time_array history data of bar opening time for the specified symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyTime(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   int              start_pos,        // start position
   int              count,            // data count to copy
   datetime         time_array[]      // target array to copy open times
   );
```

Call by the start date and the number of required elements

```
int  CopyTime(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   datetime         time_array[]      // target array to copy  open times
   );
```

Call by the start and end dates of a required time interval

```
int  CopyTime(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time,       // stop date and time
   datetime         time_array[]     // target array to copy open times
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time corresponding to the last element to copy.

time_array[]

[out] Array of datetime type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

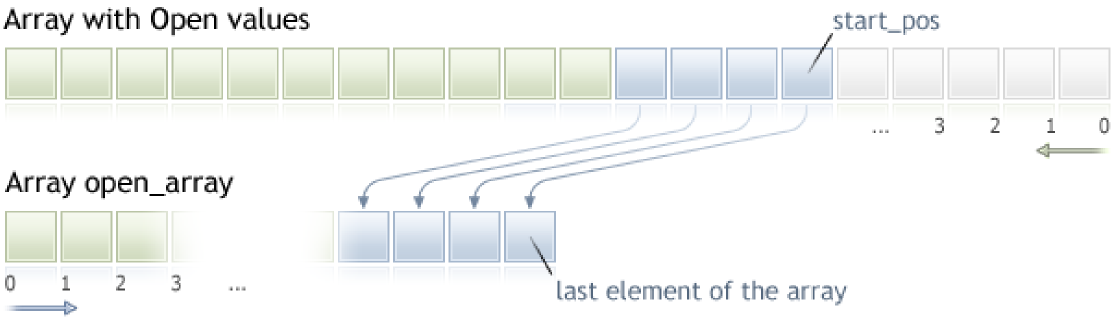
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See a detailed example of requesting history data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyOpen

The function gets into open_array the history data of bar open prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyOpen(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   int              start_pos,       // start position
   int              count,           // data count to copy
   double           open_array[]     // target array to copy open prices
   );
```

Call by the start date and the number of required elements

```
int  CopyOpen(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   int              count,           // data count to copy
   double           open_array[]     // target array for bar open prices
   );
```

Call by the start and end dates of a required time interval

```
int  CopyOpen(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time,       // stop date and time
   double           open_array[]     // target array for bar open values
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] The start time for the last element to copy.

open_array[]

[out] Array of double type.

Return Value

Returns the number of element in the array or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

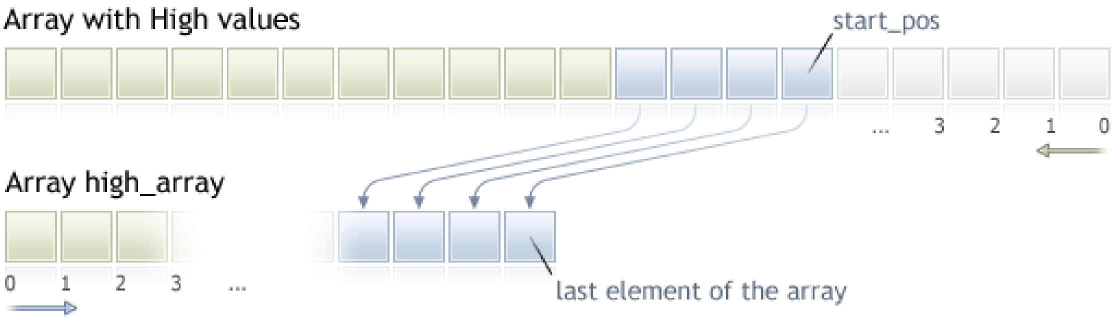
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See a detailed example of requesting history data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyHigh

The function gets into high_array the history data of highest bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyHigh(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   double           high_array[]       // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyHigh(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   double           high_array[]       // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyHigh(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   double           high_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

high_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An example for output of the High[i] and Low[i]"
#property description "for a random chosen bars"

double High[],Low[];
//+------------------------------------------------------------------+
//| Get Low for specified bar index                                  |
//+------------------------------------------------------------------+
double iLow(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double low=0;
   ArraySetAsSeries(Low,true);
   int copied=CopyLow(symbol,timeframe,0,Bars(symbol,timeframe),Low);
   if(copied>0 && index<copied) low=Low[index];
   return(low);
  }
//+------------------------------------------------------------------+
//| Get the High for specified bar index                             |
//+------------------------------------------------------------------+
double iHigh(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double high=0;
   ArraySetAsSeries(High,true);
   int copied=CopyHigh(symbol,timeframe,0,Bars(symbol,timeframe),High);
   if(copied>0 && index<copied) high=High[index];
   return(high);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
```

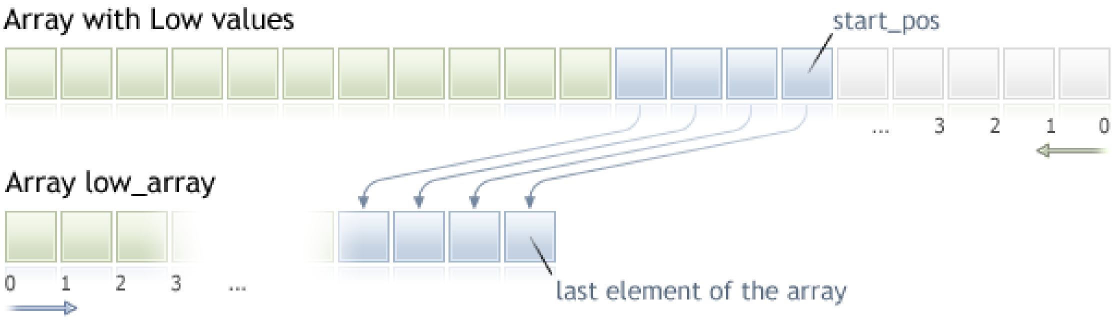
```
//--- on every tick we output the High and Low values for the bar with index,
//--- that is equal to the second, on which tick arrived
   datetime t=TimeCurrent();
   int sec=t%60;
   printf("High[%d] = %G  Low[%d] = %G",
          sec,iHigh(Symbol(),0,sec),
          sec,iLow(Symbol(),0,sec));
  }
```

See a detailed example of requesting history data in the Methods of Object Binding section. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyLow

The function gets into low_array the history data of minimal bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyLow(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   int              start_pos,        // start position
   int              count,            // data count to copy
   double           low_array[]       // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyLow(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   double           low_array[]       // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyLow(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   double           low_array[]       // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] Bar time, corresponding to the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

low_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See a detailed example of requesting history data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

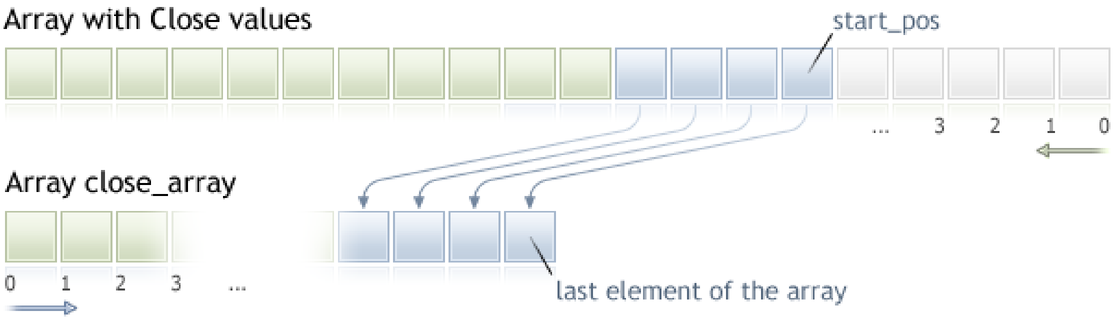
- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

See also

CopyHigh

CopyClose

The function gets into close_array the history data of bar close prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyClose(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   double           close_array[]      // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyClose(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   double           close_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyClose(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   datetime         stop_time,         // stop date and time
   double           close_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

close_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

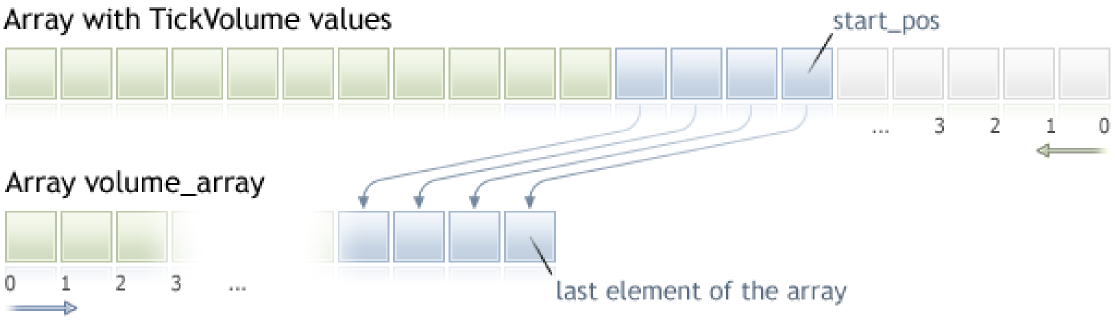
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See a detailed example of history data requesting in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyTickVolume

The function gets into volume_array the history data of tick volumes for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyTickVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   int              start_pos,        // start position
   int              count,            // data count to copy
   long             volume_array[]    // target array for tick volumes
   );
```

Call by the start date and the number of required elements

```
int  CopyTickVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   long             volume_array[]    // target array for tick volumes
   );
```

Call by the start and end dates of a required time interval

```
int  CopyTickVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   long             volume_array[]    // target array for tick volumes
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

volume_array[]

[out] Array of long type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

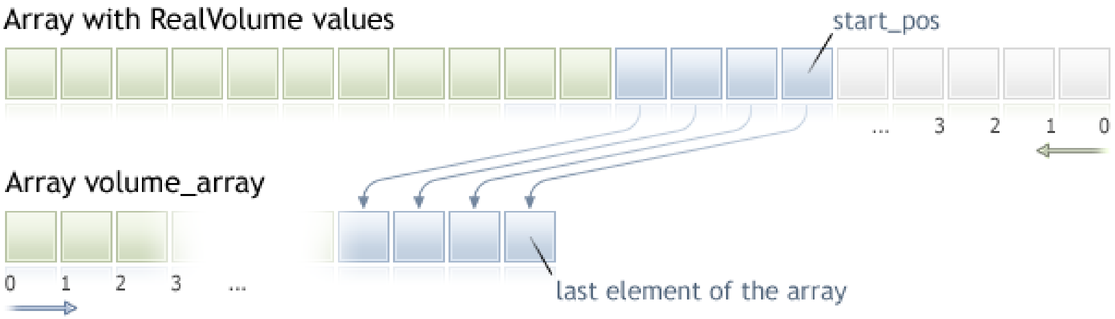
```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot TickVolume
#property indicator_label1  "TickVolume"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  C'143,188,139'
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=3000;
//--- indicator buffers
double         TickVolumeBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,TickVolumeBuffer,INDICATOR_DATA);
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                 const int prev_calculated,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      long timeseries[];
      ArraySetAsSeries(timeseries,true);
      int prices=CopyTickVolume(Symbol(),0,0,bars,timeseries);
      for(int i=0;i<rates_total-prices;i++) TickVolumeBuffer[i]=0.0;
      for(int i=0;i<prices;i++) TickVolumeBuffer[rates_total-1-i]=timeseries[prices-1-
      Print("We have received the following number of TickVolume values: "+prices);
     }
   else
     {
      long timeseries[];
      int prices=CopyTickVolume(Symbol(),0,0,1,timeseries);
      TickVolumeBuffer[rates_total-1]=timeseries[0];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See a detailed example of history data requesting in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyRealVolume

The function gets into volume_array the history data of trade volumes for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyRealVolume(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   long             volume_array[]     // target array for volumes values
   );
```

Call by the start date and the number of required elements

```
int  CopyRealVolume(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   long             volume_array[]     // target array for volumes values
   );
```

Call by the start and end dates of a required time interval

```
int  CopyRealVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   long             volume_array[]    // target array for volumes values
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

volume_array[]

[out] Array of long type.

Return Value

Returns the copied data count or -1 in the case of error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

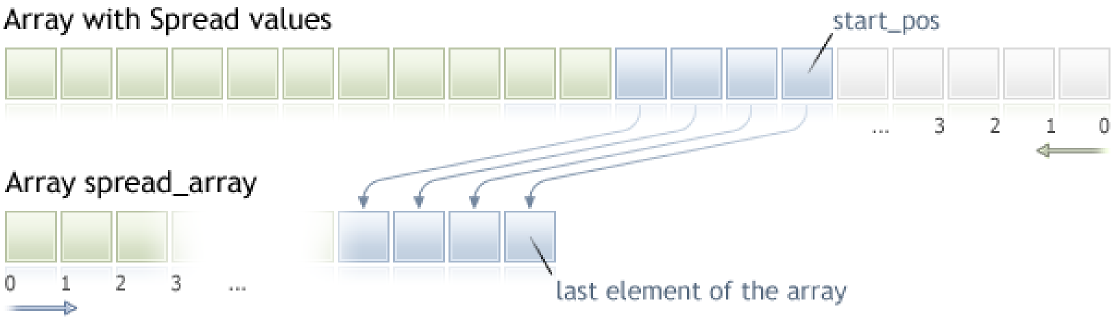
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See an example of history data requesting in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopySpread

The function gets into spread_array the history data of spread values for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopySpread(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   int              spread_array[]     // target array for spread values
   );
```

Call by the start date and the number of required elements

```
int  CopySpread(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   int              spread_array[]     // target array for spread values
   );
```

Call by the start and end dates of a required time interval

```
int  CopySpread(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   datetime         stop_time,         // stop date and time
   int              spread_array[]     // target array for spread values
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

spread_array[]

[out] Array of int type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

When requesting data from the indicator, if requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1, but the process of downloading/building will be initiated.

When requesting data from an Expert Advisor or script, downloading from the server will be initiated, if the terminal does not have these data locally, or building of a required timeseries will start, if data can be built from the local history but they are not ready yet. The function will return the amount of data that will be ready by the moment of timeout expiration, but history downloading will continue, and at the next similar request the function will return more data.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Spread
#property indicator_label1  "Spread"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=3000;
//--- indicator buffers
double         SpreadBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,SpreadBuffer,INDICATOR_DATA);
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                 const int prev_calculated,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                    const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      int spread_int[];
      ArraySetAsSeries(spread_int,true);
      int spreads=CopySpread(Symbol(),0,0,bars,spread_int);
      Print("We have received the following number of Spread values: ",spreads);
      for (int i=0;i<spreads;i++)
      {
      SpreadBuffer[rates_total-1-i]=spread_int[i];
      if(i<=30) Print("spread["+i+"] = ",spread_int[i]);
      }
     }
   else
     {
      double Ask,Bid;
      Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
      Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
      Comment("Ask = ",Ask,"  Bid = ",Bid);
      SpreadBuffer[rates_total-1]=(Ask-Bid)/Point();
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See an example of history data requesting in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyTicks

The function receives ticks in the MqlTick format into ticks_array. In this case, ticks are indexed from the past to the present, i.e. the 0 indexed tick is the oldest one in the array. For tick analysis, check the *flags* field, which shows what exactly has changed in the tick.

```
int  CopyTicks(
   string           symbol_name,           // Symbol name
   MqlTick&         ticks_array[],         // Tick receiving array
   uint             flags=COPY_TICKS_ALL,  // The flag that determines the type of re
   ulong            from=0,                // The date from which you want to request
   uint             count=0                // The number of ticks that you want to re
   );
```

Parameters

symbol_name

[in] Symbol.

ticks_array

[out] An array of the MqlTick type for receiving ticks.

flags

[in] A flag to define the type of the requested ticks. COPY_TICKS_INFO – ticks with Bid and/or Ask changes, COPY_TICKS_TRADE – ticks with changes in Last and Volume, COPY_TICKS_ALL – all ticks. For any type of request, the values of the previous tick are added to the remaining fields of the MqlTick structure.

from

[in] The date from which you want to request ticks. In milliseconds since 1970.01.01. If *from*=0, the last *count* ticks will be returned.

count

[in] The number of requested ticks. If the 'from' and 'count' parameters are not specified, all available recent ticks (but not more than 2000) will be written to ticks_array[].

Returned value

The number of copied tick or -1 in case of an error.

Note

The CopyTicks() function allows requesting and analyzing all received ticks. The first call of CopyTicks() initiates synchronization of the symbol's tick database stored on the hard disk. If the local database does not provide all the requested ticks, then missing ticks will be automatically downloaded from the trade server. Ticks beginning with the ***from*** date specified in CopyTicks() till the current moment will be synchronized. After that, all ticks arriving for this symbol will be added to the tick database thus keeping it in the synchronized state.

If the *from* and *count* parameters are not specified, all available recent ticks (but not more than 2000) will be written to *ticks_array[]*. The *flags* parameter allows specifying the type of required ticks.

COPY_TICKS_INFO – ticks with Bid and/or Ask price changes are returned. Data of other fields will also be added. For example, if only the Bid has changed, the *ask* and *volume* fields will be filled with last known values. To find out exactly what has changed, analyze the *flags* field, which will have the value of TICK_FLAG_BID and/or TICK_FLAG_ASK. If a tick has zero values of the Bid and Ask prices, and the flags show that these data have changed (*flags*=TICK_FLAG_BID|TICK_FLAG_ASK), this means that the order book (Market Depth) is empty. In other words, there are no buy and sell orders.

COPY_TICKS_TRADE – ticks with the Last price and volume changes are returned. Data of other fields will also be added, i.e. last known values of Bid and Ask will be specified in the appropriate fields. To find out exactly what has changed, analyze the *flags* field, which will have the TICK_FLAG_LAST and TICK_FLAG_VOLUME value.

COPY_TICKS_ALL – all ticks with any change are returned. Unchanged fields will be filled with last known values.

Call of CopyTicks() with the COPY_TICKS_ALL flag immediately returns all ticks from the request interval, while calls in other modes require some time to process and select ticks, therefore they do not provide significant speed advantage.

When requesting ticks (either COPY_TICKS_INFO or COPY_TICKS_TRADE), every tick contains full price information as of the time of the tick (*bid, ask, last* and *volume*). This feature is provided for an easier analysis of the trade state at the time of each tick, so there is no need to request a deep tick history and search for the values of other fields.

In indicators, the CopyTicks() function returns the result: when called from an indicator, CopyTick() immediately returns all available ticks of a symbol, and will launch synchronization of the tick database, if available data is not enough. All indicators in one symbol operate in one common thread, so the indicator cannot wait for the completion of synchronization. After synchronization, CopyTicks() will return all requested ticks during the next call. In indicators, the OnCalculate() function is called after the arrival of each tick.

CopyTicks() can wait for the result for 45 seconds in Expert Advisors and scripts: as distinct from indicators, every Expert Advisor and script operate in a separate thread, and therefore can wait 45 seconds till the completion of synchronization. If the required amount of ticks fails to be synchronized during this time, CopyTicks() will return available ticks by timeout and will continue synchronization. OnTick() in Expert Advisor is not a handler of every tick, it only notifies an Expert Advisor about changes in the market. It can be a batch of changes: the terminal can simultaneously make a few ticks, but OnTick() will be called only once to notify the EA of the latest market state.

The rate of data return: the terminal stores in the fast access cache 4,096 last ticks for each instrument (65,536 ticks for symbols with a running Market Depth). If requested ticks for the current trading session are beyond the cache, CopyTicks() calls the ticks stored in the terminal memory. These requests require more time for execution. The slowest requests are those requesting ticks for other days, since the data is read from the disk in this case.

Example:

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- Requesting 100 million ticks to be sure we receive the entire tick history
input int      getticks=100000000; // The number of required ticks
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int     attempts=0;     // Count of attempts
   bool    success=false;  // The flag of a successful copying of ticks
   MqlTick tick_array[];   // Tick receiving array
   MqlTick lasttick;       // To receive last tick data
   SymbolInfoTick(_Symbol,lasttick);
//--- Make 3 attempts to receive ticks
   while(attempts<3)
     {
      //--- Measuring start time before receiving the ticks
      uint start=GetTickCount();
//--- Requesting the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
      int received=CopyTicks(_Symbol,tick_array,COPY_TICKS_ALL,1,getticks);
      if(received!=-1)
        {
         //--- Showing information about the number of ticks and spent time
         PrintFormat("%s: received %d ticks in %d ms",_Symbol,received,GetTickCount()-
         //--- If the tick history is synchronized, the error code is equal to zero
         if(GetLastError()==0)
           {
            success=true;
            break;
           }
         else
            PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d
            _Symbol,received,GetTickCount()-start,_LastError);
        }
      //--- Counting attempts
      attempts++;
      //--- A one-second pause to wait for the end of synchronization of the tick data
      Sleep(1000);
     }
//--- Receiving the requested ticks from the beginning of the tick history failed in t
   if(!success)
     {
      PrintFormat("Error! Failed to receive %d ticks of %s in three attempts",getticks
      return;
     }
   int ticks=ArraySize(tick_array);
//--- Showing the time of the first tick in the array
   datetime firstticktime=tick_array[ticks-1].time;
   PrintFormat("Last tick time = %s.%03I64u",
               TimeToString(firstticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_ar
//--- Show the time of the last tick in the array
```

```
   datetime lastticktime=tick_array[0].time;
   PrintFormat("First tick time = %s.%03I64u",
               TimeToString(lastticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_ar

//---
   MqlDateTime today;
   datetime current_time=TimeCurrent();
   TimeToStruct(current_time,today);
   PrintFormat("current_time=%s",TimeToString(current_time));
   today.hour=0;
   today.min=0;
   today.sec=0;
   datetime startday=StructToTime(today);
   datetime endday=startday+24*60*60;
   if((ticks=CopyTicksRange(_Symbol,tick_array,COPY_TICKS_ALL,startday*1000,endday*10
     {
      PrintFormat("CopyTicksRange(%s,tick_array,COPY_TICKS_ALL,%s,%s) failed, error %
                  _Symbol,TimeToString(startday),TimeToString(endday),GetLastError());
      return;
     }
   ticks=MathMax(100,ticks);
//--- Showing the first 100 ticks of the last day
   int counter=0;
   for(int i=0;i<ticks;i++)
     {
      datetime time=tick_array[i].time;
      if((time>=startday) && (time<endday) && counter<100)
        {
         counter++;
         PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
        }
     }
//--- Showing the first 100 deals of the last day
   counter=0;
   for(int i=0;i<ticks;i++)
     {
      datetime time=tick_array[i].time;
      if((time>=startday) && (time<endday) && counter<100)
        {
         if(((tick_array[i].flags&TICK_FLAG_BUY)==TICK_FLAG_BUY) || ((tick_array[i].f
           {
            counter++;
            PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
           }
        }
     }
  }
//+------------------------------------------------------------------+
//| Returns the string description of a tick                         |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03d ",
                            TimeToString(tick.time),tick.time_msc%1000);
//--- Checking flags
   bool buy_tick=((tick.flags&TICK_FLAG_BUY)==TICK_FLAG_BUY);
   bool sell_tick=((tick.flags&TICK_FLAG_SELL)==TICK_FLAG_SELL);
   bool ask_tick=((tick.flags&TICK_FLAG_ASK)==TICK_FLAG_ASK);
   bool bid_tick=((tick.flags&TICK_FLAG_BID)==TICK_FLAG_BID);
   bool last_tick=((tick.flags&TICK_FLAG_LAST)==TICK_FLAG_LAST);
   bool volume_tick=((tick.flags&TICK_FLAG_VOLUME)==TICK_FLAG_VOLUME);
//--- Checking trading flags in a tick first
   if(buy_tick || sell_tick)
     {
      //--- Forming an output for the trading tick
      desc=desc+(buy_tick?StringFormat("Buy Tick: Last=%G Volume=%d ",tick.last,tick.v
      desc=desc+(sell_tick?StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last,tick
      desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
      desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
      desc=desc+"(Trade tick)";
     }
   else
     {
      //--- Form a different output for an info tick
      desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
      desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
      desc=desc+(last_tick?StringFormat("Last=%G ",tick.last):"");
      desc=desc+(volume_tick?StringFormat("Volume=%d ",tick.volume):"");
      desc=desc+"(Info tick)";
     }
//--- Returning tick description
   return desc;
  }
//+------------------------------------------------------------------+
/* Example of the output
Si-12.16: received 11048387 ticks in 4937 ms
Last tick time = 2016.09.26 18:32:59.775
First tick time = 2015.06.18 09:45:01.000
1.  2016.09.26 09:45.249 Ask=65370 Bid=65370 (Info tick)
2.  2016.09.26 09:47.420 Ask=65370 Bid=65370 (Info tick)
3.  2016.09.26 09:50.893 Ask=65370 Bid=65370 (Info tick)
4.  2016.09.26 09:51.827 Ask=65370 Bid=65370 (Info tick)
5.  2016.09.26 09:53.810 Ask=65370 Bid=65370 (Info tick)
6.  2016.09.26 09:54.491 Ask=65370 Bid=65370 (Info tick)
7.  2016.09.26 09:55.913 Ask=65370 Bid=65370 (Info tick)
8.  2016.09.26 09:59.350 Ask=65370 Bid=65370 (Info tick)
9.  2016.09.26 09:59.678 Bid=65370 (Info tick)
10. 2016.09.26 10:00.000 Sell Tick: Last=65367 Volume=3 (Trade tick)
```

```
11. 2016.09.26 10:00.000 Sell Tick: Last=65335 Volume=45 (Trade tick)
12. 2016.09.26 10:00.000 Sell Tick: Last=65334 Volume=95 (Trade tick)
13. 2016.09.26 10:00.191 Sell Tick: Last=65319 Volume=1 (Trade tick)
14. 2016.09.26 10:00.191 Sell Tick: Last=65317 Volume=1 (Trade tick)
15. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=1 (Trade tick)
16. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=10 (Trade tick)
17. 2016.09.26 10:00.191 Sell Tick: Last=65315 Volume=5 (Trade tick)
18. 2016.09.26 10:00.191 Sell Tick: Last=65313 Volume=3 (Trade tick)
19. 2016.09.26 10:00.191 Sell Tick: Last=65307 Volume=25 (Trade tick)
20. 2016.09.26 10:00.191 Sell Tick: Last=65304 Volume=1 (Trade tick)
21. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=1 (Trade tick)
22. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=10 (Trade tick)
23. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=5 (Trade tick)
24. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=1 (Trade tick)
25. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=6 (Trade tick)
26. 2016.09.26 10:00.191 Sell Tick: Last=65299 Volume=1 (Trade tick)
27. 2016.09.26 10:00.191 Bid=65370 (Info tick)
28. 2016.09.26 10:00.232 Ask=65297 (Info tick)
29. 2016.09.26 10:00.276 Sell Tick: Last=65291 Volume=31 (Trade tick)
30. 2016.09.26 10:00.276 Sell Tick: Last=65290 Volume=1 (Trade tick)
*/
```

See also

SymbolInfoTick, Structure for Current Prices, OnTick()

CopyTicksRange

The function receives ticks in the MqlTick format within the specified date range to ticks_array. Indexing goes from the past to the present meaning that a tick with the index 0 is the oldest one in the array. For tick analysis, check the *flags* field, which shows what exactly has changed.

```
int  CopyTicksRange(
   const string      symbol_name,           // symbol name
   MqlTick&          ticks_array[],         // tick receiving array
   uint              flags=COPY_TICKS_ALL,  // flag that defines the type of the ticks
   ulong             from_msc=0,            // date, starting from which ticks are requ
   ulong             to_msc=0               // date, up to which ticks are requested
   );
```

Parameters

symbol_name

[in] Symbol.

ticks_array

[out] MqlTick static or dynamic array for receiving ticks. If the static array cannot hold all the ticks from the requested time interval, the maximum possible amount of ticks is received. In this case, the function generates the error ERR_HISTORY_SMALL_BUFFER (4407) .

flags

[in] A flag to define the type of the requested ticks. COPY_TICKS_INFO – ticks with Bid and/or Ask changes, COPY_TICKS_TRADE – ticks with changes in Last and Volume, COPY_TICKS_ALL – all ticks. For any type of request, the values of the previous tick are added to the remaining fields of the MqlTick structure.

from_msc

[in] The date, from which you want to request ticks. In milliseconds since 1970.01.01. If the *from_msc* parameter is not specified, ticks from the beginning of the history are sent. Ticks with the time >= from_msc are sent.

to_msc

[in] The date, up to which you want to request ticks. In milliseconds since 01.01.1970. Ticks with the time <= to_msc are sent. If the *to_msc* parameter is not specified, all ticks up to the end of the history are sent.

Return Value

The number of copied tick or -1 in case of an error. GetLastError() is able to return the following errors:

- ERR_HISTORY_TIMEOUT – ticks synchronization waiting time is up, the function has sent all it had.
- ERR_HISTORY_SMALL_BUFFER – static buffer is too small. Only the amount the array can store has been sent.
- ERR_NOT_ENOUGH_MEMORY – insufficient memory for receiving a history from the specified range to the dynamic tick array. Failed to allocate enough memory for the tick array.

Note

The CopyTicksRange() function is used for requesting ticks strictly from a specified range, for example, from a certain day in history. At the same time, CopyTicks() allows specifying only a start date, for example – receive all ticks from the beginning of the month till the current moment.

See also

SymbolInfoTick, Structure for Current Prices, OnTick, CopyTicks

iBars

Returns the number of bars of a corresponding symbol and period, available in history.

```
int  iBars(
   const string           symbol,          // Symbol
   ENUM_TIMEFRAMES        timeframe        // Period
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

Return Value

The number of bars of a corresponding symbol and period, available in history, but no more than allowed by the "Max bars in chart" parameter in platform settings.

Example:

```
  Print("Bar count on the 'EURUSD,H1' is ",iBars("EURUSD",PERIOD_H1));
```

See also

Bars

iBarShift

Search bar by time. The function returns the index of the bar corresponding to the specified time.

```
int  iBarShift(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   datetime            time,            // Time
   bool                exact=false      // Mode
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. PERIOD_CURRENT means the current chart period.

time

[in] Time value to search for.

exact=false

[in] A return value, in case the bar with the specified time is not found. If *exact=false*, *iBarShift* returns the index of the nearest bar, the Open time of which is less than the specified time (*time_open*<*time*). If such a bar is not found (history before the specified time is not available), then the function returns -1. If *exact=true*, iBarShift does not search for a nearest bar but immediately returns -1.

Return Value

The index of the bar corresponding to the specified time. If the bar corresponding to the specified time is not found (there is a gap in the history), the function returns -1 or the index of the nearest bar (depending on the '*exact*' parameter).

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- The date is on Sunday
   datetime time=D'2002.04.25 12:00';
   string symbol="GBPUSD";
   ENUM_TIMEFRAMES tf=PERIOD_H1;
   bool exact=false;
//--- If there is no bar at the specified time, iBarShift will return the index of the
   int bar_index=iBarShift(symbol,tf,time,exact);
//--- Check the error code after the call of iBarShift()
   int error=GetLastError();
   if(error!=0)
     {
      PrintFormat("iBarShift(): GetLastError=%d - The requested date %s "+
                  "for %s %s is not found in the available history",
                  error,TimeToString(time),symbol,EnumToString(tf));
      return;
     }
//--- The iBarShift() function was executed successfully, return a result for exact=fa
   PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),
               DayOfWeek(time),bar_index,string(exact));
   datetime bar_time=iTime(symbol,tf,bar_index);
   PrintFormat("Time of bar #%d is %s (%s)",
               bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
//--- Request the index of the bar with the specified time; if there is no bar -1 wil
   exact=true;
   bar_index=iBarShift(symbol,tf,time,exact);
//--- The iBarShift() function was executed successfully, return a result for exact=tr
   PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time)
               ,DayOfWeek(time),bar_index,string(exact));
  }
//+------------------------------------------------------------------+
//| Returns the name of the day of the week                          |
//+------------------------------------------------------------------+
string DayOfWeek(const datetime time)
  {
   MqlDateTime dt;
   string day="";
   TimeToStruct(time,dt);
   switch(dt.day_of_week)
     {
      case 0: day=EnumToString(SUNDAY);
      break;
      case 1: day=EnumToString(MONDAY);
      break;
      case 2: day=EnumToString(TUESDAY);
      break;
      case 3: day=EnumToString(WEDNESDAY);
      break;
      case 4: day=EnumToString(THURSDAY);
      break;
      case 5: day=EnumToString(FRIDAY);
      break;
      default:day=EnumToString(SATURDAY);
      break;
     }
//---
   return day;
  }
//+------------------------------------------------------------------+
/* Execution result
   1. GBPUSD PERIOD_H1 2018.06.10 12:00(SUNDAY): bar index is 64 (exact=false)
   Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
   2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY):bar index is -1 (exact=true)
*/
```

iClose

Returns the Close price of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
double  iClose(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The Close price of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
   }
```

See also

CopyClose, CopyRates

iHigh

Returns the High price of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
double  iHigh(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The High price of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
  }
```

See also

[CopyHigh](), [CopyRates]()

iHighest

Returns the index of the highest value found on the corresponding chart (shift relative to the current bar).

```
int  iHighest(
   const string        symbol,               // Symbol
   ENUM_TIMEFRAMES     timeframe,            // Period
   ENUM_SERIESMODE     type,                 // Timeseries identifier
   int                 count=WHOLE_ARRAY,    // Number of elements
   int                 start=0               // Index
  );
```

Parameters

symbol

[in] The symbol, on which the search will be performed. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

type

[in] The identifier of the timeseries, in which the search will be performed. Can be equal to any value from ENUM_SERIESMODE.

count=WHOLE_ARRAY

[in] The number of elements in the timeseries (from the current bar towards index increasing direction), among which the search should be performed.

start=0

[in] The index (shift relative to the current bar) of the initial bar, from which search for the highest value begins. Negative values are ignored and replaced with a zero value.

Return Value

The index of the highest value found on the corresponding chart (shift relative to the current bar) or -1 in case of an error. For error details, call the GetLastError() function.

Example:

```
   double val;
//--- Calculation of the highest Close value among 20 consecutive bars
//--- From index 4 to index 23 inclusive, on the current timeframe
   int val_index=iHighest(NULL,0,MODE_CLOSE,20,4);
   if(val_index!=-1)
      val=High[val_index];
   else
      PrintFormat("iHighest() call error. Error code=%d",GetLastError());
```

iLow

Returns the Low price of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
double  iLow(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The Low price of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
  }
```

See also

CopyLow, CopyRates

iLowest

Returns the index of the smallest value found on the corresponding chart (shift relative to the current bar).

```
int  iLowest(
   const string         symbol,               // Symbol
   ENUM_TIMEFRAMES      timeframe,            // Period
   ENUM_SERIESMODE      type,                 // Timeseries identifier
   int                  count=WHOLE_ARRAY,    // Number of elements
   int                  start=0               // Index
  );
```

Parameters

symbol

[in] The symbol, on which the search will be performed. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

type

[in] The identifier of the timeseries, in which the search will be performed. Can be equal to any value from ENUM_SERIESMODE.

count=WHOLE_ARRAY

[in] The number of elements in the timeseries (from the current bar towards index increasing direction), among which the search should be performed.

start=0

[in] The index (shift relative to the current bar) of the initial bar, from which search for the lowest value begins. Negative values are ignored and replaced with a zero value.

Return Value

The index of the lowest value found on the corresponding chart (shift relative to the current bar) or -1 in case of an error. For error details, call the GetLastError() function.

Example:

```
   double val;
//--- Search for a bar with the lowest value of the real volume among 15 consecutive
//--- From index 10 to index 24 inclusive, on the current timeframe
   int val_index=iLowest(NULL,0,MODE_REAL_VOLUME,15,10);
   if(val_index!=-1)
      val=Low[val_index];
   else
      PrintFormat("iLowest() call error. Error code=%d",GetLastError());
```

iOpen

Returns the Open price of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
double  iOpen(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The Open price of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "   ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "   ,DoubleToString(open,Digits()),"\n",
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
   }
```

See also

CopyOpen, CopyRates

iTime

Returns the opening time of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
datetime  iTime(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The opening time of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- The date is on Sunday
   datetime time=D'2018.06.10 12:00';
   string symbol="GBPUSD";
   ENUM_TIMEFRAMES tf=PERIOD_H1;
   bool exact=false;
//--- there is no bar at the specified time, iBarShift will return the index of the ne
   int bar_index=iBarShift(symbol,tf,time,exact);
   PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,st
   datetime bar_time=iTime(symbol,tf,bar_index);
   PrintFormat("Time of bar #%d is %s (%s)",
               bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
//PrintFormat(iTime(symbol,tf,bar_index));
//--- Request the index of the bar with the specified time; but there is no bar, retu
   exact=true;
   bar_index=iBarShift(symbol,tf,time,exact);
   PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
               symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,st
  }
//+------------------------------------------------------------------+
//| Returns the name of the day of the week                          |
//+------------------------------------------------------------------+
string DayOfWeek(const datetime time)
  {
   MqlDateTime dt;
   string day="";
   TimeToStruct(time,dt);
   switch(dt.day_of_week)
     {
      case 0: day=EnumToString(SUNDAY);
      break;
      case 1: day=EnumToString(MONDAY);
      break;
      case 2: day=EnumToString(TUESDAY);
      break;
      case 3: day=EnumToString(WEDNESDAY);
      break;
      case 4: day=EnumToString(THURSDAY);
      break;
      case 5: day=EnumToString(FRIDAY);
      break;
      default:day=EnumToString(SATURDAY);
      break;
     }
//---
   return day;
  }
/* The result:
   1. GBPUSD PERIOD_H1 2018.06.10 12:00(SUNDAY): bar index is 64 (exact=false)
   Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
   2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY):bar index is -1 (exact=true)
*/
```

See also

CopyTime, CopyRates

iTickVolume

Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
long  iTickVolume(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
  }
```

See also

CopyTickVolume, CopyRates

iRealVolume

Returns the real volume of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
long  iRealVolume(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The real volume of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
   }
```

See also

CopyRealVolume, CopyRates

iVolume

Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
long  iVolume(
   const string        symbol,          // Symbol
   ENUM_TIMEFRAMES     timeframe,       // Period
   int                 shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "   ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "   ,DoubleToString(open,Digits()),"\n",
            "High: "   ,DoubleToString(high,Digits()),"\n",
            "Low: "    ,DoubleToString(low,Digits()),"\n",
            "Close: "  ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "   ,IntegerToString(bars),"\n"
            );
  }
```

See also

CopyTickVolume, CopyRates

iSpread

Returns the spread value of the bar (indicated by the 'shift' parameter) on the corresponding chart.

```
long  iSpread(
   const string          symbol,          // Symbol
   ENUM_TIMEFRAMES       timeframe,       // Period
   int                   shift            // Shift
   );
```

Parameters

symbol

[in] The symbol name of the financial instrument. NULL means the current symbol.

timeframe

[in] Period. It can be one of the values of the ENUM_TIMEFRAMES enumeration. 0 means the current chart period.

shift

[in] The index of the received value from the timeseries (backward shift by specified number of bars relative to the current bar).

Return Value

The Spread value of the bar (indicated by the 'shift' parameter) on the corresponding chart or 0 in case of an error. For error details, call the GetLastError() function.

Note

The function always returns actual data. For this purpose it performs a request to the timeseries for the specified symbol/period during each call. This means that if there is no ready data during the first function call, some time may be taken to prepare the result.

The function does not store previous calls results, and there is no local cache for quick value return.

Example:

```
input int shift=0;
//+------------------------------------------------------------------+
//| Function-event handler "tick"                                    |
//+------------------------------------------------------------------+
void OnTick()
  {
   datetime time  = iTime(Symbol(),Period(),shift);
   double   open  = iOpen(Symbol(),Period(),shift);
   double   high  = iHigh(Symbol(),Period(),shift);
   double   low   = iLow(Symbol(),Period(),shift);
   double   close = iClose(NULL,PERIOD_CURRENT,shift);
   long     volume= iVolume(Symbol(),0,shift);
   int      bars  = iBars(NULL,0);

   Comment(Symbol(),",",EnumToString(Period()),"\n",
            "Time: "  ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
            "Open: "  ,DoubleToString(open,Digits()),"\n",
            "High: "  ,DoubleToString(high,Digits()),"\n",
            "Low: "   ,DoubleToString(low,Digits()),"\n",
            "Close: " ,DoubleToString(close,Digits()),"\n",
            "Volume: ",IntegerToString(volume),"\n",
            "Bars: "  ,IntegerToString(bars),"\n"
            );
  }
```

See also

CopySpread, CopyRates

Custom symbols

Functions for creating and editing the custom symbol properties.

When connecting the terminal to a certain trade server, a user is able to work with time series of the financial symbols provided by a broker. Available financial symbols are displayed as a list in the Market Watch window. A separate group of functions allows receiving data on the symbol properties, trading sessions and market depth updates.

The group of functions described in this section allows creating custom symbols. To do this, users are able to apply the trade server's existing symbols, text files or external data sources.

Function	Action
CustomSymbolCreate	Create a custom symbol with the specified name in the specified group
CustomSymbolDelete	Delete a custom symbol with the specified name
CustomSymbolSetInteger	Set the integer type property value for a custom symbol
CustomSymbolSetDouble	Set the real type property value for a custom symbol
CustomSymbolSetString	Set the string type property value for a custom symbol
CustomSymbolSetMarginRate	Set the margin rates depending on the order type and direction for a custom symbol
CustomSymbolSetSessionQuote	Set the start and end time of the specified quotation session for the specified symbol and week day
CustomSymbolSetSessionTrade	Set the start and end time of the specified trading session for the specified symbol and week day
CustomRatesDelete	Delete all bars from the price history of the custom symbol in the specified time interval
CustomRatesReplace	Fully replace the price history of the custom symbol within the specified time interval with the data from the MqlRates type array
CustomRatesUpdate	Add missing bars to the custom symbol history and replace existing data with the ones from the MqlRates type array
CustomTicksAdd	Adds data from an array of the MqlTick type to the price history of a custom symbol. The custom symbol must be selected in the Market Watch window
CustomTicksDelete	Delete all ticks from the price history of the custom symbol in the specified time interval
CustomTicksReplace	Fully replace the price history of the custom symbol within the specified time interval with the data from the MqlTick type array
CustomBookAdd	Passes the status of the Depth of Market for a custom symbol

CustomSymbolCreate

Creates a custom symbol with the specified name in the specified group.

```
bool  CustomSymbolCreate(
   const string     symbol_name,          // custom symbol name
   const string     symbol_path="",       // name of a group a symbol is to be created
   const string     symbol_origin=NULL    // name of a symbol used as a basis to create
   );
```

Parameters

symbol_name

[in] Custom symbol name. It should not contain groups or subgroups the symbol is located in.

symbol_path=""

[in] The group name a symbol is located in.

symbol_origin=NULL

[in] Name of a symbol the properties of a created custom symbol are to be copied from. After creating a custom symbol, any property value can be changed to a necessary one using the appropriate functions.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

All custom symbols are created in the special Custom section. If a group name is not specified (the *symbol_path* parameter in the CustomSymbolCreate function contains an empty string or NULL), a custom symbol is generated in the Custom section root. Here we can draw an analogy with the file system, where groups and subgroups can be viewed as folders and subfolders

The symbol and group names may only contain Latin letters without punctuation, spaces or special characters (may only contain ".", "_", "&" and "#"). It is not recommended to use characters <, >, :, ", /, |, ?, *.

The custom symbol name should be unique regardless of a group name it is created in. If a symbol with the same name already exists, the CustomSymbolCreate() function returns 'false', while the subsequent GetLastError() call returns the error 5300 (ERR_NOT_CUSTOM_SYMBOL) or 5304 (ERR_CUSTOM_SYMBOL_EXIST).

The length of the symbol name should not exceed 31 characters. Otherwise, CustomSymbolCreate() returns 'false' and the error 5302 – ERR_CUSTOM_SYMBOL_NAME_LONG is activated.

The *symbol_path* parameter can be set in two ways:

- only a group name without a name of the custom symbol, for example – "CFD\\Metals". It is best to use this option to avoid errors.
- or <group> name + groups separator "\\"+<custom symbol name>, for example – "CFD\\Metals\\Platinum". In this case, the group name should end with the exact name of the custom symbol. In case of a mismatch, the custom symbol is still created, but not in the intended group. For example, if *symbol_path*="CFD\\Metals\\Platinum" and *symbol_name*="**p**latinum" (register error),

then a custom symbol named "platinum" is created in the "Custom\CFD\Metals\Platinum" group. The SymbolInfoGetString("platinum",SYMBOL_PATH) function returns the "Custom\CFD\Metals\Platinum\platinum" value.

Note that the SYMBOL_PATH property returns the path with the symbol name at the end. Therefore, it cannot be copied without changes if you want to create a custom symbol in the exact same group. In this case, it is necessary to cut the symbol name in order not to get the result described above.

If a non-existent symbol is set as the *symbol_origin* parameter, then the custom symbol is created empty as if the *symbol_origin* parameter is not set. The error 4301 - ERR_MARKET_UNKNOWN_SYMBOL is activated in that case.

The *symbol_path* parameter length should not exceed 127 characters considering "Custom\\", "\\" groups separators and the symbol name if it is specified at the end.

Example:

```
//+------------------------------------------------------------------+
//|                                           CustomSymbolCreate.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"        // name of the group, in which a symbo
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // name of a symbol a custom one is t

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- if failed to create a custom symbol, inform of that in the journal
   if(!CustomSymbolCreate(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL_ORIGI
     {
      Print("CustomSymbolCreate() failed. Error ", GetLastError());
      return;
     }

//--- check the existence of the created symbol and get the group it was created in
   bool   custom= false;
   bool   exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   string path  = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_PATH);

//--- print the result of creating the symbol and the name of the group (specified an
   PrintFormat("Custom symbol '%s' created\n"+
                "Symbol '%s' is exist: %s\n"+
                "Symbol '%s' is custom: %s\n"+
                "Path specified in the settings: '%s'\n"+
                "Path returned from the 'SYMBOL_PATH' property: '%s'",
                CUSTOM_SYMBOL_NAME,
                CUSTOM_SYMBOL_NAME, (string)exist,
                CUSTOM_SYMBOL_NAME, (string)custom,
                CUSTOM_SYMBOL_PATH,
                path);

//--- wait two seconds and delete the created symbol with the resulting message in the
   Sleep(2000);
   ResetLastError();
   bool deleted = CustomSymbolDelete(CUSTOM_SYMBOL_NAME);
   Print(deleted ? StringFormat("Custom symbol '%s' removed", CUSTOM_SYMBOL_NAME) : St
   /*
   result:
   Custom symbol 'EURUSD.C' created
   Symbol 'EURUSD.C' is exist: true
   Symbol 'EURUSD.C' is custom: true
   Path specified in the settings: 'Forex'
   Path returned from the 'SYMBOL_PATH' property: 'Custom\Forex\EURUSD.C'
   Custom symbol 'EURUSD.C' removed
   */
  }
```

See also

SymbolName, SymbolSelect, CustomSymbolDelete

CustomSymbolDelete

Deletes a custom symbol with the specified name.

```
bool  CustomSymbolDelete(
   const string     symbol_name          // custom symbol name
   );
```

Parameters

symbol

[in] Custom symbol name. It should not match the name of an already existing symbol.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

The custom symbol displayed in the Market Watch or the one a chart is opened for cannot be deleted.

Example:

```
//+------------------------------------------------------------------+
//|                                           CustomSymbolDelete.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"        // name of the group, in which a symbo
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // name of a symbol a custom one is to

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- if failed to create a custom symbol, inform of that in the journal
   if(!CustomSymbolCreate(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOL_ORIGIN
     {
      Print("CustomSymbolCreate() failed. Error ", GetLastError());
      return;
     }
```

```
//--- check the existence of the created symbol and print the result in the journal
   bool custom= false;
   bool exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   PrintFormat("Custom symbol '%s' exists: %s", CUSTOM_SYMBOL_NAME, (string)exist);

//--- wait two seconds and delete the created symbol with the resulting message in the
   Sleep(2000);
   ResetLastError();
   bool deleted = CustomSymbolDelete(CUSTOM_SYMBOL_NAME);
   Print(deleted ? StringFormat("Custom symbol '%s' removed", CUSTOM_SYMBOL_NAME) : St

//--- check the existence of the created symbol and print the result in the journal
   exist = SymbolExist(CUSTOM_SYMBOL_NAME, custom);
   PrintFormat("Custom symbol '%s' exists: %s", CUSTOM_SYMBOL_NAME, (string)exist);
   /*
   result:
   Custom symbol 'EURUSD.C' exists: true
   Custom symbol 'EURUSD.C' removed
   Custom symbol 'EURUSD.C' exists: false
   */
  }
```

See also

SymbolName, SymbolSelect, CustomSymbolCreate

CustomSymbolSetInteger

Sets the integer type property value for a custom symbol.

```
bool  CustomSymbolSetInteger(
   const string              symbol_name,       // symbol name
   ENUM_SYMBOL_INFO_INTEGER  property_id,       // property ID
   long                      property_value     // property value
   );
```

Parameters

symbol_name

[in] Custom symbol name.

property_id

[in] Symbol property ID. The value can be one of the values of the ENUM_SYMBOL_INFO_INTEGER enumeration.

property_value

[in] A long type variable containing the property value.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

The minute and tick history of the custom symbol is completely removed if any of these properties is changed in the symbol specification:

- SYMBOL_CHART_MODE – price type for constructing bars (Bid or Last)
- SYMBOL_DIGITS – number of digits after the decimal point to display the price

After deleting the custom symbol history, the terminal attempts to create a new history using the updated properties. The same happens when the custom symbol properties are changed manually.

Example:

```
//+------------------------------------------------------------------+
//|                                       CustomSymbolSetInteger.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"        // name of the group, in which a symbo
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // name of a symbol a custom one is to
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol ha
   if(create!=0 && create!=5304)
      return;

//--- get and print in the journal the properties of the symbol the custom one is bas
//--- (trading mode, Stop order installation level and trading operations freezing di
   ENUM_SYMBOL_TRADE_EXECUTION origin_exe_mode = (ENUM_SYMBOL_TRADE_EXECUTION)SymbolI
   int origin_stops_level = (int)SymbolInfoInteger(CUSTOM_SYMBOL_ORIGIN, SYMBOL_TRADE_
   int origin_freeze_level= (int)SymbolInfoInteger(CUSTOM_SYMBOL_ORIGIN, SYMBOL_TRADE_

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Deal execution mode: %s\n  Stops Level: %d\n  Freeze Level: %d",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               StringSubstr(EnumToString(origin_exe_mode), 23), origin_stops_level, o

//--- set other values for the custom symbol properties
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_EXEMODE, SYMBOL_TRAD
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_STOPS_LEVEL, 10);
   res &=CustomSymbolSetInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_FREEZE_LEVEL, 3);

//--- if there was an error when setting any of the properties, display an appropriat
   if(!res)
      Print("CustomSymbolSetInteger() failed. Error ", GetLastError());

//--- get and print in the journal the modified custom symbol properties
//--- (trading mode, Stop order installation level and trading operations freezing di
   ENUM_SYMBOL_TRADE_EXECUTION custom_exe_mode = (ENUM_SYMBOL_TRADE_EXECUTION)SymbolI
   int custom_stops_level = (int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_S
   int custom_freeze_level= (int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_TRADE_F

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Deal execution mode: %s\n  Stops Level: %d\n  Freeze Level: %d",
               CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_ORIGIN,
               StringSubstr(EnumToString(custom_exe_mode), 23), custom_stops_level, c

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Deal execution mode: INSTANT
     Stops Level: 0
     Freeze Level: 0
   Custom symbol 'EURUSD.C' based on 'EURUSD':
     Deal execution mode: MARKET
     Stops Level: 10
     Freeze Level: 3
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name,
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

SymbolInfoInteger

CustomSymbolSetDouble

Sets the real type property value for a custom symbol.

```
bool  CustomSymbolSetDouble(
   const string              symbol_name,       // symbol name
   ENUM_SYMBOL_INFO_DOUBLE   property_id,       // property ID
   double                    property_value     // property value
   );
```

Parameters

symbol_name

[in] Custom symbol name.

property_id

[in] Symbol property ID. The value can be one of the values of the ENUM_SYMBOL_INFO_DOUBLE enumeration.

property_value

[in] A double type variable containing the property value.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

The minute and tick history of the custom symbol is completely removed if any of these properties is changed in the symbol specification:

- SYMBOL_POINT – one point value
- SYMBOL_TRADE_TICK_SIZE – value of a tick that specifies the minimum allowable price change
- SYMBOL_TRADE_TICK_VALUE – one-tick price change value for a profitable position

After deleting the custom symbol history, the terminal attempts to create a new history using the updated properties. The same happens when the custom symbol properties are changed manually.

Example:

```
//+------------------------------------------------------------------+
//|                                        CustomSymbolSetDouble.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"   // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"         // name of the group, in which a symbo
```

```
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // name of a symbol a custom one is to

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOI

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get and print in the journal the properties of the symbol the custom one is base
//--- (minimum volume, maximum volume, minimum volume change step for deal execution)
   double origin_vol_min = SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_MIN);
   double origin_vol_max = SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_MAX);
   double origin_vol_step= SymbolInfoDouble(CUSTOM_SYMBOL_ORIGIN, SYMBOL_VOLUME_STEP);

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Volume Min: %.2f\n  Volume Max: %.2f\n  Volume Step: %.2f",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               origin_vol_min, origin_vol_max, origin_vol_step);

//--- set other values for the custom symbol properties
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MIN, 0.1);
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MAX, 1000);
   res &=CustomSymbolSetDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_STEP, 0.1);

//--- if there was an error when setting any of the properties, display an appropriate
   if(!res)
      Print("CustomSymbolSetDouble() failed. Error ", GetLastError());

//--- get and print in the journal the modified custom symbol properties
//--- (minimum volume, maximum volume, minimum volume change step for deal execution)
   double custom_vol_min = SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MIN);
   double custom_vol_max = SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_MAX);
   double custom_vol_step= SymbolInfoDouble(CUSTOM_SYMBOL_NAME, SYMBOL_VOLUME_STEP);

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Volume Min: %.2f\n  Volume Max: %.2f\n  Volume Step: %.2f",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               custom_vol_min, custom_vol_max, custom_vol_step);

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Volume Min: 0.01
     Volume Max: 500.00
     Volume Step: 0.01
   Custom symbol 'EURUSD' based on 'EURUSD.C':
     Volume Min: 0.10
     Volume Max: 1000.00
     Volume Step: 0.10
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

SymbolInfoDouble

CustomSymbolSetString

Sets the string type property value for a custom symbol.

```
bool  CustomSymbolSetString(
   const string              symbol_name,      // symbol name
   ENUM_SYMBOL_INFO_STRING   property_id,      // property ID
   string                    property_value    // property value
   );
```

Parameters

symbol_name

[in] Custom symbol name.

property_id

[in] Symbol property ID. The value can be one of the values of the ENUM_SYMBOL_INFO_STRING enumeration.

property_value

[in] A string type variable containing the property value.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

The minute and tick history of the custom symbol is completely removed if the SYMBOL_FORMULA property (setting the equation for the custom symbol price construction) is changed in the symbol specification. After deleting the custom symbol history, the terminal attempts to create a new history using the new equation. The same happens when the custom symbol equation is changed manually.

Example:

```
//+------------------------------------------------------------------+
//|                                      CustomSymbolSetString.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"   // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"         // name of the group, in which a symbo
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()        // name of a symbol a custom one is to

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol ha
   if(create!=0 && create!=5304)
      return;

//--- get and print in the journal the properties of the symbol the custom one is base
//--- (minimum volume, maximum volume, minimum volume change step for deal execution)
   string origin_basis    = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_BASIS);
   string origin_category = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_CATEGORY);
   string origin_formula  = SymbolInfoString(CUSTOM_SYMBOL_ORIGIN, SYMBOL_FORMULA);

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Basis: %s\n  Category: %s\n  Formula: %s",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               origin_basis, origin_category, origin_formula);

//--- set other values for the custom symbol properties
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_BASIS, CUSTOM_SYMBOL_ORIGIN)
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_CATEGORY, "FX");
   res &=CustomSymbolSetString(CUSTOM_SYMBOL_NAME, SYMBOL_FORMULA, ("1.0 / "+CUSTOM_S

//--- if there was an error when setting any of the properties, display an appropriate
   if(!res)
      Print("CustomSymbolSetString() failed. Error ", GetLastError());

//--- get and print in the journal the modified custom symbol properties
//--- (minimum volume, maximum volume, minimum volume change step for deal execution)
   string custom_basis    = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_BASIS);
   string custom_category = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_CATEGORY);
   string custom_formula  = SymbolInfoString(CUSTOM_SYMBOL_NAME, SYMBOL_FORMULA);

   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Basis: %s\n  Category: %s\n  Formula: %s",
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               custom_basis, custom_category, custom_formula);

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e

//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
```

```
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Basis: 
     Category: 
     Formula: 
   Custom symbol 'EURUSD' based on 'EURUSD.C':
     Basis: EURUSD
     Category: FX
     Formula: 1.0 / EURUSD
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
```

```
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

SymbolInfoString

CustomSymbolSetMarginRate

Sets the margin rates depending on the order type and direction for a custom symbol.

```
bool  CustomSymbolSetMarginRate(
   const string       symbol_name,              // symbol name
   ENUM_ORDER_TYPE    order_type,               // order type
   double             initial_margin_rate,      // initial margin rate
   double             maintenance_margin_rate   // maintenance margin rate
   );
```

Parameters

symbol_name

[in] Custom symbol name.

order_type

[in] Order type.

initial_margin_rate

[in] A double type variable with an initial margin rate. Initial margin is a security deposit for 1 lot deal in the appropriate direction. Multiplying the rate by the initial margin, we receive the amount of funds to be reserved on the account when placing an order of the specified type.

maintenance_margin_rate

[in] A double type variable with a maintenance margin rate. Maintenance margin is a minimum amount for maintaining an open position of 1 lot in the appropriate direction. Multiplying the rate by the maintenance margin, we receive the amount of funds to be reserved on the account after an order of the specified type is activated.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//|                                    CustomSymbolSetMarginRate.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"  // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"        // name of the group, in which a symbo
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()       // name of a symbol a custom one is to
```

```
#define   INITIAL_MARGIN_RATE       1.5         // initial margin rate
#define   MAINTENANCE_MARGIN_RATE   1.5         // maintenance margin rate

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get and print in the journal the properties of the symbol the custom one is base
//--- (initial and maintenance margin rates for Buy and Sell orders)
   double initial_margin_rate_buy=0;
   double maintenance_margin_rate_buy=0;
   double initial_margin_rate_sell=0;
   double maintenance_margin_rate_sell=0;

   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_ORIGIN, ORDER_TYPE_BUY, initial_margin_rate_
      return;
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_ORIGIN, ORDER_TYPE_SELL,initial_margin_rate_
      return;

   PrintFormat("The '%s' symbol from which the custom '%s' was created:\n"+
               "  Buy order initial margin rate: %f\n  Buy order maintenance margin ra
               "  Sell order initial margin rate: %f\n  Sell order maintenance margin
               CUSTOM_SYMBOL_ORIGIN, CUSTOM_SYMBOL_NAME,
               initial_margin_rate_buy, maintenance_margin_rate_buy, initial_margin_ra

//--- set other values for the custom symbol properties
   ResetLastError();
   bool res=true;
   res &=CustomSymbolSetMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_BUY, INITIAL_MARGIN_
   res &=CustomSymbolSetMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_SELL,INITIAL_MARGIN_

//--- if there was an error when setting any of the properties, display an appropriate
   if(!res)
      Print("CustomSymbolSetMarginRate() failed. Error ", GetLastError());

//--- get and print in the journal the modified custom symbol properties
//--- (initial and maintenance margin rates for Buy and Sell orders)
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_BUY, initial_margin_rate_bu
      return;
   if(!GetSymbolMarginRate(CUSTOM_SYMBOL_NAME, ORDER_TYPE_SELL,initial_margin_rate_se
      return;
   PrintFormat("Custom symbol '%s' based on '%s':\n"+
               "  Buy order initial margin rate: %f\n  Buy order maintenance margin ra
               "  Sell order initial margin rate: %f\n  Sell order maintenance margin 
               CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_ORIGIN,
               initial_margin_rate_buy, maintenance_margin_rate_buy, initial_margin_ra

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex

//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   The 'EURUSD' symbol from which the custom 'EURUSD.C' was created:
     Buy order initial margin rate: 1.000000
     Buy order maintenance margin rate: 0.000000
     Sell order initial margin rate: 1.000000
     Sell order maintenance margin rate: 0.000000
   Custom symbol 'EURUSD.C' based on 'EURUSD':
     Buy order initial margin rate: 1.500000
     Buy order maintenance margin rate: 1.500000
     Sell order initial margin rate: 1.500000
     Sell order maintenance margin rate: 1.500000
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Return margin ratios                                             |
//+------------------------------------------------------------------+
bool GetSymbolMarginRate(const string symbol, const ENUM_ORDER_TYPE order_type, double
  {
   ResetLastError();
   if(!SymbolInfoMarginRate(symbol, order_type, initial_margin_rate, maintenance_marg
     {
      PrintFormat("%s: SymbolInfoMarginRate(%s, %s) failed. Error %d",__FUNCTION__, sy
      return false;
     }
   return true;
  }
```

See also

SymbolInfoMarginRate

CustomSymbolSetSessionQuote

Sets the start and end time of the specified quotation session for the specified symbol and week day.

```
bool  CustomSymbolSetSessionQuote(
   const string       symbol_name,          // symbol name
   ENUM_DAY_OF_WEEK   day_of_week,          // week day
   uint               session_index,        // session index
   datetime           from,                 // session start time
   datetime           to                    // session end time
   );
```

Parameters

symbol_name

[in] Custom symbol name.

ENUM_DAY_OF_WEEK

[in] Week day, value from the ENUM_DAY_OF_WEEK enumeration.

uint

[in] Index of the session, for which start and end times are to be set. Session indexing starts from 0.

from

[in] Session start time in seconds from 00:00, data value in the variable is ignored.

to

[in] Session end time in seconds from 00:00, data value in the variable is ignored.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

If the session with the specified *session_index* already exists, the function simply edits the beginning and end of the session.

If zero start and end parameters have been passed for the session (*from=0* and *to=0*), the appropriate session with the *session_index* is deleted, while the session indexing is shifted downwards.

Sessions can be added only sequentially. In other words, you can add *session_index=1* only if the session with the index 0 already exists. If this rule is broken, a new session is not created, while the function itself returns 'false'.

Example:

```
//+------------------------------------------------------------------+
//|                                    CustomSymbolSetSessionQuote.mq5 |
//|                                   Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"           // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"                 // name of the group, in whic
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()                // name of a symbol a custom

#define   SESSION_0_FROM         D'1970.01.01 00:15:00'  // session 0 start time
#define   SESSION_0_TO           D'1970.01.01 11:59:00'  // session 0 end time
#define   SESSION_1_FROM         D'1970.01.01 12:15:00'  // session 1 start time
#define   SESSION_1_TO           D'1970.01.01 23:59:00'  // session 1 end time

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol ha
   if(create!=0 && create!=5304)
      return;

//--- print the header with the base symbol and session index and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of
   for(int session=0; session<2; session++)
     {
      PrintFormat("Quote session %d of '%s' symbol from which the custom '%s' was crea
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_ORIGIN, (ENUM_DAY_OF_WEEK)day_of_we
     }

//--- in a loop by two sessions
   bool res=true;
   for(int session=0; session<2; session++)
     {
      datetime from = SESSION_0_FROM;
      datetime to   = SESSION_0_TO;
      if(session>0)
        {
         from = SESSION_1_FROM;
         to   = SESSION_1_TO;
        }
      //--- set the quote sessions time for a custom symbol of each day of the week
      ResetLastError();
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         res &=CustomSymbolSetSessionQuote(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_
     }

//--- if there was an error when setting any of the sessions, display an appropriate 
   if(!res)
      Print("CustomSymbolSetSessionQuote() failed. Error ", GetLastError());

//--- print the header with the custom symbol and session index and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of 
   for(int session=0; session<2; session++)
     {
      PrintFormat("Quote session %d of custom symbol '%s' based on '%s'", session, CU
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_of_wee
     }

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Quote session 0 of 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     00:15 - 23:55
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   Quote session 1 of 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     Session not set
   - Tuesday    Session not set
   - Wednesday  Session not set
   - Thursday   Session not set
   - Friday     Session not set
   Quote session 0 of custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     00:15 - 11:59
   - Tuesday    00:15 - 11:59
   - Wednesday  00:15 - 11:59
   - Thursday   00:15 - 11:59
   - Friday     00:15 - 11:59
   Quote session 1 of custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     12:15 - 23:59
   - Tuesday    12:15 - 23:59
   - Wednesday  12:15 - 23:59
   - Thursday   12:15 - 23:59
   - Friday     12:15 - 23:59
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
```

```
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Send the start and end times of the specified quote session      |
//| for the specified symbol and day of the week to the journal      |
//+------------------------------------------------------------------+
void SymbolInfoSessionQuotePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- declare variables to record the beginning and end of the quote session
   datetime date_from;  // session start time
   datetime date_to;    // session end time

//--- create the week day name from the enumeration constant
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- get data from the quotation session by symbol and day of the week
   if(!SymbolInfoSessionQuote(symbol, day_of_week, session_index, date_from, date_to))
     {
      int err=GetLastError();
      string message=(err==4307 ? StringFormat("- %-10s Session not set", week_day) :
                      StringFormat("SymbolInfoSessionQuote(%s, %s, session %d) failed.
      Print(message);
      return;
     }

//--- send data for the specified quote session to the journal
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Ti
  }
```

See also

SymbolInfoSessionQuote, Symbol info, TimeToStruct, Date structure

CustomSymbolSetSessionTrade

Sets the start and end time of the specified trading session for the specified symbol and week day.

```
bool  CustomSymbolSetSessionTrade(
   const string       symbol_name,         // symbol name
   ENUM_DAY_OF_WEEK   day_of_week,         // week day
   uint               session_index,       // session index
   datetime           from,                // session start time
   datetime           to                   // session end time
   );
```

Parameters

symbol_name

[in] Custom symbol name.

ENUM_DAY_OF_WEEK

[in] Week day, value from the ENUM_DAY_OF_WEEK enumeration.

uint

[in] Index of the session, for which start and end times are to be set. Session indexing starts from 0.

from

[in] Session start time in seconds from 00:00, data value in the variable is ignored.

to

[in] Session end time in seconds from 00:00, data value in the variable is ignored.

Return Value

true – success, otherwise – false. To get information about the error, call the GetLastError() function.

Note

If the session with the specified *session_index* already exists, the function simply edits the beginning and end of the session.

If zero start and end parameters have been passed for the session (*from=0* and *to=0*), the appropriate session with the *session_index* is deleted, while the session indexing is shifted downwards.

Sessions can be added only sequentially. In other words, you can add *session_index=1* only if the session with the index 0 already exists. If this rule is broken, a new session is not created, while the function itself returns 'false'.

Example:

```
//+------------------------------------------------------------------+
//|                                  CustomSymbolSetSessionTrade.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"           // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"                 // name of the group, in whic
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()                // name of a symbol a custom

#define   SESSION_0_FROM         D'1970.01.01 00:15:00'  // session 0 start time
#define   SESSION_0_TO           D'1970.01.01 11:59:00'  // session 0 end time
#define   SESSION_1_FROM         D'1970.01.01 12:15:00'  // session 1 start time
#define   SESSION_1_TO           D'1970.01.01 23:59:00'  // session 1 end time

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- print the header with the base symbol and session index and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of
   for(int session=0; session<2; session++)
     {
      PrintFormat("Trade session %d of the '%s' symbol from which the custom '%s' was
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_ORIGIN, (ENUM_DAY_OF_WEEK)day_of_we
     }

//--- in a loop by two sessions
   bool res=true;
   for(int session=0; session<2; session++)
     {
      datetime from = SESSION_0_FROM;
      datetime to   = SESSION_0_TO;
      if(session>0)
        {
         from = SESSION_1_FROM;
         to   = SESSION_1_TO;
        }
      //--- set the quote sessions time for a custom symbol of each day of the week
      ResetLastError();
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         res &=CustomSymbolSetSessionQuote(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_o
     }

//--- if there was an error when setting any of the sessions, display an appropriate m
   if(!res)
      Print("CustomSymbolSetSessionTrade() failed. Error ", GetLastError());

//--- print the header with the custom symbol and session index and
//--- in a loop by day of the week from Mon to Fri, print the start and end times of e
   for(int session=0; session<2; session++)
     {
      PrintFormat("Trade session %d of a custom symbol '%s' based on '%s'", session, C
      for(int day_of_week=MONDAY; day_of_week<SATURDAY; day_of_week++)
         SymbolInfoSessionQuotePrint(CUSTOM_SYMBOL_NAME, (ENUM_DAY_OF_WEEK)day_of_week
     }

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Trade session 0 of the 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     00:15 - 23:55
   - Tuesday    00:15 - 23:55
   - Wednesday  00:15 - 23:55
   - Thursday   00:15 - 23:55
   - Friday     00:15 - 23:55
   Trade session 1 of the 'EURUSD' symbol from which the custom 'EURUSD.C' was created
   - Monday     Session not set
   - Tuesday    Session not set
   - Wednesday  Session not set
   - Thursday   Session not set
   - Friday     Session not set
   Trade session 0 of a custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     00:15 - 11:59
   - Tuesday    00:15 - 11:59
   - Wednesday  00:15 - 11:59
   - Thursday   00:15 - 11:59
   - Friday     00:15 - 11:59
   Trade session 1 of a custom symbol 'EURUSD.C' based on 'EURUSD'
   - Monday     12:15 - 23:59
   - Tuesday    12:15 - 23:59
   - Wednesday  12:15 - 23:59
   - Thursday   12:15 - 23:59
   - Friday     12:15 - 23:59
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
```

```
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Send the start and end times of the specified quote session      |
//| for the specified symbol and day of the week to the journal      |
//+------------------------------------------------------------------+
void SymbolInfoSessionTradePrint(const string symbol, const ENUM_DAY_OF_WEEK day_of_we
  {
//--- declare variables to record the beginning and end of the quote session
   datetime date_from;  // session start time
   datetime date_to;    // session end time

//--- create the week day name from the enumeration constant
   string week_day=EnumToString(day_of_week);
   if(week_day.Lower())
      week_day.SetChar(0, ushort(week_day.GetChar(0)-32));

//--- get data from the quotation session by symbol and day of the week
   if(!SymbolInfoSessionTrade(symbol, day_of_week, session_index, date_from, date_to))
     {
      int err=GetLastError();
      string message=(err==4307 ? StringFormat("- %-10s Session not set", week_day) :
                      StringFormat("SymbolInfoSessionTrade(%s, %s, session %d) failed
      Print(message);
      return;
     }

//--- send data for the specified trading session to the journal
   PrintFormat("- %-10s %s - %s", week_day, TimeToString(date_from, TIME_MINUTES), Ti
  }
```

See also

SymbolInfoSessionTrade, Symbol info, TimeToStruct, Date structure

CustomRatesDelete

Deletes all bars from the price history of the custom symbol in the specified time interval.

```
int  CustomRatesDelete(
   const string      symbol,        // symbol name
   datetime          from,          // start date
   datetime          to             // end date
   );
```

Parameters

symbol

[in] Custom symbol name.

from

[in] Time of the first bar in the price history within the specified range to be removed.

to

[in] Time of the last bar in the price history within the specified range to be removed.

Return Value

Number of deleted bars or -1 in case of an error.

Example:

```
//+------------------------------------------------------------------+
//|                                          CustomRatesDelete.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"           // name of the group, in which a s
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // name of a symbol a custom one i

#define   DATARATES_COUNT        4                 // number of bars sent to the jour

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO
//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get the number of standard symbol bars
   int bars=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);

//--- get the data of all bars of the standard symbol minute timeframe into the MqlRat
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_OR
      return;
     }

//--- set the copied data to the minute history of the custom symbol
   ResetLastError();
   if(CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates)<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- after updating the historical data, get the number of custom symbol bars
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- get the data of all bars of the custom symbol minute timeframe into the MqlRates
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- print the last DATARATES_COUNT bars of the custom symbol minute history in the j
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- delete the two penultimate data bars in the custom symbol minute history
   datetime time_from= rates[bars-3].time;
   datetime time_to  = rates[bars-2].time;
   ResetLastError();
   int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, time_from, time_to);
   if(deleted<0)
     {
      PrintFormat("CustomRatesDelete(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- after deleting two bars of historical data, get the number of custom symbol bars
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- get the data of all remaining bars of the custom symbol minute timeframe again
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- print the last DATARATES_COUNT bars of the updated custom symbol minute history
   PrintFormat("\nLast %d bars after applying CustomRatesDelete() with %d deleted bars
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully 

         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully

         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Last 4 bars of the custom symbol's minute history:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 20:53:00 1.07341 1.07347 1.07336 1.07343            38        0
```

```
    [1] 2024.06.18 20:54:00 1.07344 1.07354 1.07344 1.07353            21         0
    [2] 2024.06.18 20:55:00 1.07353 1.07362 1.07351 1.07356            32         0
    [3] 2024.06.18 20:56:00 1.07356 1.07358 1.07352 1.07354            24         0

   Last 4 bars after applying CustomRatesDelete() with 2 deleted bars:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
    [0] 2024.06.18 20:51:00 1.07357 1.07358 1.07347 1.07349            25         0
    [1] 2024.06.18 20:52:00 1.07349 1.07350 1.07336 1.07341            31         0
    [2] 2024.06.18 20:53:00 1.07341 1.07347 1.07336 1.07343            38         0
    [3] 2024.06.18 20:56:00 1.07356 1.07358 1.07352 1.07354            24         0
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastError
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

[CustomRatesReplace](), [CustomRatesUpdate](), [CopyRates]()

CustomRatesReplace

Fully replaces the price history of the custom symbol within the specified time interval with the data from the MqlRates type array.

```
int  CustomRatesReplace(
   const string      symbol,             // symbol name
   datetime          from,               // start date
   datetime          to,                 // end date
   const MqlRates&   rates[],            // array for the data to be applied to a custo
   uint              count=WHOLE_ARRAY   // number of the rates[] array elements to be
   );
```

Parameters

symbol

[in] Custom symbol name.

from

[in] Time of the first bar in the price history within the specified range to be updated.

to

[in] Time of the last bar in the price history within the specified range to be updated.

rates[]

[in] Array of the MqlRates type history data for M1.

count=WHOLE_ARRAY

[in] Number of the *rates[]* array elements to be used for replacement. WHOLE_ARRAY means that all *rates[]* array elements should be used for replacement.

Return Value

Number of updated bars or -1 in case of an error.

Note

If the bar from the *rates[]* array goes beyond the specified range, it is ignored. If such a bar is already present in the price history and enters the given range, it is replaced. All other bars in the current price history outside the specified range remain unchanged. The *rates[]* array data should be correct regarding OHLC prices, while the bars opening time should correspond to the M1 timeframe.

Example:

```
//+------------------------------------------------------------------+
//|                                           CustomRatesReplace.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
```

```
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"           // name of the group, in which a sy
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // name of a symbol a custom one is

#define   DATARATES_COUNT        4                 // number of bars sent to the journ

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOI

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get the number of standard symbol bars
   int bars=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);

//--- get the data of all bars of the standard symbol minute timeframe into the MqlRat
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_OI
      return;
     }

//--- set the copied data to the minute history of the custom symbol
   ResetLastError();
   if(CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates)<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- after updating the historical data, get the number of custom symbol bars
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- get the data of all bars of the custom symbol minute timeframe into the MqlRates
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- print the last DATARATES_COUNT bars of the custom symbol minute history in the
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- change the two penultimate data bars in the custom symbol minute history
   datetime time_from= rates[bars-3].time;
   datetime time_to  = rates[bars-2].time;

//--- make all prices of the two penultimate bars equal to the open prices of these ba
   rates[bars-3].high=rates[bars-3].open;
   rates[bars-3].low=rates[bars-3].open;
   rates[bars-3].close=rates[bars-3].open;

   rates[bars-2].high=rates[bars-2].open;
   rates[bars-2].low=rates[bars-2].open;
   rates[bars-2].close=rates[bars-2].open;

//--- replace existing bars with data from the modified 'rates' array
   ResetLastError();
   int replaced=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(replaced<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- after changing two bars of historical data, get the number of custom symbol bars
   bars=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);

//--- get the data of all bars of the custom symbol minute timeframe again
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars, rates)!=bars)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- print the last DATARATES_COUNT bars of the updated custom symbol minute history
   PrintFormat("\nLast %d bars after applying CustomRatesUpdate() with %d replaced bar
   ArrayPrint(rates, digits, NULL, bars-DATARATES_COUNT, DATARATES_COUNT);

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully 

         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully

         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Last 4 bars of the custom symbol's minute history:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.07.29 13:37:00 1.08394 1.08396 1.08388 1.08390            16        1
   [1] 2024.07.29 13:38:00 1.08389 1.08400 1.08389 1.08398            35        1
   [2] 2024.07.29 13:39:00 1.08398 1.08410 1.08394 1.08410            29        1
   [3] 2024.07.29 13:40:00 1.08409 1.08414 1.08408 1.08414            14        1

   Last 4 bars after applying CustomRatesUpdate() with 250820 replaced bars:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.07.29 13:37:00 1.08394 1.08396 1.08388 1.08390            16        1
   [1] 2024.07.29 13:38:00 1.08389 1.08389 1.08389 1.08389            35        1
   [2] 2024.07.29 13:39:00 1.08398 1.08398 1.08398 1.08398            29        1
   [3] 2024.07.29 13:40:00 1.08409 1.08414 1.08408 1.08414            14        1
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in 
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

CustomRatesDelete, CustomRatesUpdate, CopyRates

CustomRatesUpdate

Adds missing bars to the custom symbol history and replaces existing data with the ones from the MqlRates type array.

```
int  CustomRatesUpdate(
   const string      symbol,             // custom symbol name
   const MqlRates&   rates[],            // array for the data to be applied to a custo
   uint              count=WHOLE_ARRAY   // number of the rates[] array elements to be
   );
```

Parameters

symbol

[in] Custom symbol name.

rates[]

[in] Array of the MqlRates type history data for M1.

count=WHOLE_ARRAY

[in] Number of the *rates[]* array elements to be used for update. WHOLE_ARRAY means that all *rates[]* array elements should be used.

Return Value

Number of updated bars or -1 in case of an error.

Note

If there is no bar from the *rates[]* array in the current custom symbol history, it is added. If such a bar already exists, it is replaced. All other bars in the current price history remain unchanged. The *rates[]* array data should be correct regarding OHLC prices, while the bars opening time should correspond to the M1 timeframe.

Example:

```
//+------------------------------------------------------------------+
//|                                          CustomRatesUpdate.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"           // name of the group, in which a sy
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // name of a symbol a custom one is

#define   DATARATES_COUNT        4                 // number of bars sent to the journ
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol ha
   if(create!=0 && create!=5304)
      return;

//--- get and print in the journal the number of standard symbol bars
   int bars_origin=Bars(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1);
   PrintFormat("The symbol '%s' from which the custom '%s' was created has %d bars of

//--- get and print in the journal the number of custom symbol bars
   int bars_custom=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);
   PrintFormat("Custom symbol '%s' created from symbol '%s' has %d bars of minute hist

//--- get the data of all bars of the standard symbol minute timeframe into the MqlRat
   MqlRates rates[]={};
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_ORIGIN, PERIOD_M1, 0, bars_origin, rates)!=bars_origin)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_O
      return;
     }

//--- set the copied data to the minute history of the custom symbol
   ResetLastError();
   int updated=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(updated<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- get and print in the journal the number of custom symbol bars after adding hist
   bars_custom=Bars(CUSTOM_SYMBOL_NAME, PERIOD_M1);
   PrintFormat("\nAfter CustomRatesUpdate(), the custom symbol '%s' has %d bars of mi

//--- get the data of all bars of the custom symbol minute timeframe into the MqlRates
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars_custom, rates)!=bars_custom)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_N
      return;
     }
```

```
//--- print the last four bars of the custom symbol minute history in the journal
   int digits=(int)SymbolInfoInteger(CUSTOM_SYMBOL_NAME, SYMBOL_DIGITS);
   PrintFormat("Last %d bars of the custom symbol's minute history:", DATARATES_COUNT)
   ArrayPrint(rates, digits, NULL, bars_custom-DATARATES_COUNT, DATARATES_COUNT);

//--- replace the data in the MqlRates array with the one calculated using the equati
   for(int i=0; i<bars_custom; i++)
     {
      rates[i].open  =(rates[i].open !=0  ? 1.0 / rates[i].open  : rates[i].open);
      rates[i].high  =(rates[i].high !=0  ? 1.0 / rates[i].high  : rates[i].high);
      rates[i].low   =(rates[i].low  !=0  ? 1.0 / rates[i].low   : rates[i].low);
      rates[i].close =(rates[i].close!=0  ? 1.0 / rates[i].close : rates[i].close);
     }

//--- set the modified data to the minute history of the custom symbol
   ResetLastError();
   updated=CustomRatesUpdate(CUSTOM_SYMBOL_NAME, rates);
   if(updated<0)
     {
      PrintFormat("CustomRatesUpdate(%s) failed. Error %d", CUSTOM_SYMBOL_NAME, GetLas
      return;
     }

//--- get the data of all bars of the custom symbol minute timeframe into the MqlRate
   ResetLastError();
   if(CopyRates(CUSTOM_SYMBOL_NAME, PERIOD_M1, 0, bars_custom, rates)!=bars_custom)
     {
      PrintFormat("CopyRates(%s, PERIOD_M1, 0, %d) failed. Error %d", CUSTOM_SYMBOL_NA
      return;
     }

//--- print the last four bars of the updated custom symbol minute history in the jou
   Print("\nLast %d bars after changing the custom symbol calculation formula:", DATA
   ArrayPrint(rates, digits, NULL, bars_custom-DATARATES_COUNT, DATARATES_COUNT);

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully
         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully

         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   The symbol 'EURUSD' from which the custom 'EURUSD.C' was created has 250488 bars o
   Custom symbol 'EURUSD.C' created from symbol 'EURUSD' has 0 bars of minute history

   After CustomRatesUpdate(), the custom symbol 'EURUSD.C' has 250488 bars of minute 
   Last 4 bars of the custom symbol's minute history:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 11:14:00 1.07235 1.07239 1.07232 1.07239            24        0
   [1] 2024.06.18 11:15:00 1.07238 1.07239 1.07232 1.07235            44        0
   [2] 2024.06.18 11:16:00 1.07234 1.07238 1.07227 1.07234            37        0
   [3] 2024.06.18 11:17:00 1.07234 1.07234 1.07217 1.07225            41        0

   Last 4 bars after changing the custom symbol calculation formula:
                    [time]  [open]  [high]   [low] [close] [tick_volume] [spread] [rea
   [0] 2024.06.18 11:14:00 0.93253 0.93250 0.93256 0.93250            24        0
   [1] 2024.06.18 11:15:00 0.93251 0.93250 0.93256 0.93253            44        0
   [2] 2024.06.18 11:16:00 0.93254 0.93251 0.93260 0.93254            37        0
   [3] 2024.06.18 11:17:00 0.93254 0.93254 0.93269 0.93262            41        0
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, 
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
```

See also

[CustomRatesReplace](), [CustomRatesDelete](), [CopyRates]()

CustomTicksAdd

Adds data from an array of the MqlTick type to the price history of a custom symbol. The custom symbol must be selected in the Market Watch window.

```
int  CustomTicksAdd(
   const string      symbol,             // Symbol name
   const MqlTick&    ticks[],            // The array with tick data that should be app
   uint              count=WHOLE_ARRAY   // number of the ticks[] array elements to be
   );
```

Parameters

symbol

[in] The name of the custom symbol.

ticks[]

[in] An array of tick data of the MqlTick type arranged in order of time from earlier data to more recent ones, i.e. ticks[k].time_msc <= ticks[n].time_msc, if k<n.

count=WHOLE_ARRAY

[in] Number of the *ticks[]* array elements to be used for adding. WHOLE_ARRAY means that all *ticks[]* array elements should be used.

Return Value

The number of added ticks or -1 in case of an error.

Further Note

The CustomTicksAdd function only works for custom symbols opened in the Market Watch window. If the symbol is not selected in Market Watch, then you should add ticks using CustomTicksReplace.

The CustomTicksAdd function allows transmitting ticks as if they are delivered from the broker's server. Data are sent to the Market Watch window instead of being written directly to the tick database. The terminal then saves ticks from the Market Watch to a database. If the amount of data transmitted during one function call is large, the behavior of the function changes in order to reduce resource usage. If more than 256 ticks are passed, data is divided into two parts. The first, i.e. the larger part is written directly to the tick database (as it is done in CustomTicksReplace). The second part containing 128 ticks is passed to the Market Watch window, from which the terminal saves the ticks to a database.

The MqlTick structure has two fields with the time value: time (the tick time in seconds) and time_msc (the tick time in milliseconds), which are counted from January 1, 1970. These fields in the added ticks are processed in the following order:

1. If ticks[k].time_msc!=0, we use it to fill the ticks[k].time field, i.e. ticks[k].time=ticks[k].time_msc/1000 (integer division) is set for the tick
2. If ticks[k].time_msc==0 and ticks[k].time!=0, time in milliseconds is obtained by multiplying by 1000, i.e. ticks[k].time_msc=ticks[k].time*1000
3. If ticks[k].time_msc==0 and ticks[k].time==0, the current trade server time up to a millisecond as of the moment of CustomTicksAdd call is written to these fields.

If the value of ticks[k].bid, ticks[k].ask, ticks[k].last or ticks[k].volume is greater than zero, a combination of appropriate flags is written to the ticks[k].flags field:

- TICK_FLAG_BID – the tick has changed the bid price
- TICK_FLAG_ASK – the tick has changed the ask price
- TICK_FLAG_LAST – the tick has changed the last deal price
- TICK_FLAG_VOLUME – the tick has changed the volume

If the value of a field is less than or equal to zero, the corresponding flag is not written to the ticks[k].flags field.

Flags TICK_FLAG_BUY and TICK_FLAG_SELL are not added to the history of a custom symbol.

Example:

```
//+------------------------------------------------------------------+
//|                                               CustomTicksAdd.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"           // name of the group, in which a sy
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // name of a symbol a custom one is

#define   DATATICKS_TO_COPY      UINT_MAX          // number of ticks copied
#define   DATATICKS_TO_PRINT     20                // number of ticks sent to the jou

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get the standard symbol tick data to the MqlTick array
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- print the time of the first and last received ticks of the standard symbol
```

```
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the standard symbol in the journal
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRINT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- add a custom symbol to the MarketWatch window
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- add the tick array data to the custom symbol price history
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", ar
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", add

//--- get the custom symbol tick data to the MqlTick array
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- print the time of the first and last received ticks of the custom symbol
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the custom symbol in the journal
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }
```

```
//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully 

         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully

         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Requested 4294967295 ticks to download tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351183943 ti
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.20 21:18:12.010

   The last 20 ticks for the standard symbol 'EURUSD':
     351183923th Tick: 2024.06.20 21:17:46.380 Bid=1.07124 (Info tick)
     351183924th Tick: 2024.06.20 21:17:47.779 Ask=1.07125 Bid=1.07125 (Info tick)
     351183925th Tick: 2024.06.20 21:17:48.584 Ask=1.07124 Bid=1.07124 (Info tick)
     351183926th Tick: 2024.06.20 21:17:49.481 Ask=1.07125 (Info tick)
     351183927th Tick: 2024.06.20 21:17:49.985 Ask=1.07122 Bid=1.07122 (Info tick)
     351183928th Tick: 2024.06.20 21:17:50.482 Ask=1.07124 Bid=1.07124 (Info tick)
     351183929th Tick: 2024.06.20 21:17:51.584 Ask=1.07123 Bid=1.07123 (Info tick)
     351183930th Tick: 2024.06.20 21:17:52.786 Ask=1.07124 Bid=1.07124 (Info tick)
     351183931th Tick: 2024.06.20 21:17:53.487 Ask=1.07125 Bid=1.07125 (Info tick)
     351183932th Tick: 2024.06.20 21:17:53.989 Ask=1.07126 Bid=1.07126 (Info tick)
     351183933th Tick: 2024.06.20 21:17:55.789 Ask=1.07125 Bid=1.07125 (Info tick)
     351183934th Tick: 2024.06.20 21:17:58.495 Ask=1.07126 Bid=1.07126 (Info tick)
     351183935th Tick: 2024.06.20 21:18:00.102 Bid=1.07126 (Info tick)
     351183936th Tick: 2024.06.20 21:18:00.698 Ask=1.07129 Bid=1.07129 (Info tick)
     351183937th Tick: 2024.06.20 21:18:03.699 Bid=1.07129 (Info tick)
     351183938th Tick: 2024.06.20 21:18:04.699 Ask=1.07128 Bid=1.07128 (Info tick)
     351183939th Tick: 2024.06.20 21:18:05.901 Ask=1.07129 Bid=1.07129 (Info tick)
     351183940th Tick: 2024.06.20 21:18:07.606 Ask=1.07128 Bid=1.07128 (Info tick)
     351183941th Tick: 2024.06.20 21:18:11.512 Ask=1.07127 Bid=1.07127 (Info tick)
     351183942th Tick: 2024.06.20 21:18:12.010 Ask=1.07126 Bid=1.07126 (Info tick)
   ...
   Start of adding 351183943 ticks to the history of the custom symbol 'EURUSD.C'
   Added 351183943 ticks to the history of the custom symbol 'EURUSD.C' in 269890 ms
   ...
   Requested 351183943 ticks to download tick history for the symbol 'EURUSD.C'
   The tick history for the 'EURUSD.C' symbol is received in the amount of 351183943 t
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.20 21:18:12.010

   The last 20 ticks for the custom symbol 'EURUSD.C':
     351183923th Tick: 2024.06.20 21:17:46.380 Ask=1.07124 Bid=1.07124 (Info tick)
     351183924th Tick: 2024.06.20 21:17:47.779 Ask=1.07125 Bid=1.07125 (Info tick)
     351183925th Tick: 2024.06.20 21:17:48.584 Ask=1.07124 Bid=1.07124 (Info tick)
     351183926th Tick: 2024.06.20 21:17:49.481 Ask=1.07125 Bid=1.07125 (Info tick)
     351183927th Tick: 2024.06.20 21:17:49.985 Ask=1.07122 Bid=1.07122 (Info tick)
     351183928th Tick: 2024.06.20 21:17:50.482 Ask=1.07124 Bid=1.07124 (Info tick)
     351183929th Tick: 2024.06.20 21:17:51.584 Ask=1.07123 Bid=1.07123 (Info tick)
     351183930th Tick: 2024.06.20 21:17:52.786 Ask=1.07124 Bid=1.07124 (Info tick)
     351183931th Tick: 2024.06.20 21:17:53.487 Ask=1.07125 Bid=1.07125 (Info tick)
     351183932th Tick: 2024.06.20 21:17:53.989 Ask=1.07126 Bid=1.07126 (Info tick)
     351183933th Tick: 2024.06.20 21:17:55.789 Ask=1.07125 Bid=1.07125 (Info tick)
     351183934th Tick: 2024.06.20 21:17:58.495 Ask=1.07126 Bid=1.07126 (Info tick)
     351183935th Tick: 2024.06.20 21:18:00.102 Ask=1.07126 Bid=1.07126 (Info tick)
     351183936th Tick: 2024.06.20 21:18:00.698 Ask=1.07129 Bid=1.07129 (Info tick)
     351183937th Tick: 2024.06.20 21:18:03.699 Ask=1.07129 Bid=1.07129 (Info tick)
     351183938th Tick: 2024.06.20 21:18:04.699 Ask=1.07128 Bid=1.07128 (Info tick)
     351183939th Tick: 2024.06.20 21:18:05.901 Ask=1.07129 Bid=1.07129 (Info tick)
     351183940th Tick: 2024.06.20 21:18:07.606 Ask=1.07128 Bid=1.07128 (Info tick)
     351183941th Tick: 2024.06.20 21:18:11.512 Ask=1.07127 Bid=1.07127 (Info tick)
     351183942th Tick: 2024.06.20 21:18:12.010 Ask=1.07126 Bid=1.07126 (Info tick)
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Get the specified number of ticks in the array                   |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- notify of the start of loading historical data
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, sy

//--- make 3 attempts to receive ticks
   int attempts=0;
   while(attempts<3)
     {
      //--- measure the start time before receiving the ticks
      uint start=GetTickCount();

      //--- request the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
      int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
      if(received!=-1)
        {
         //--- display information about the number of ticks and spent time
         PrintFormat("The tick history for the '%s' symbol is received in the amount 

         //--- if the tick history is synchronized, the error code is equal to zero -
         if(GetLastError()==0)
            return(true);

         PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms
                     symbol, received, GetTickCount()-start, GetLastError());
        }
      //--- count attempts
      attempts++;
      //--- a one-second pause to wait for the end of synchronization of the tick data
      Sleep(1000);
     }
//--- failed to copy ticks in 3 attempts
   return(false);
  }
//+------------------------------------------------------------------+
//| return the string description of a tick                          |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTE

//--- check tick flags
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)    == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)   == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)    == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)    == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)   == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- check the tick for trading flags first (there are none for CustomTicksAdd())
   if(buy_tick || sell_tick)
     {
      //--- form an output for a trading tick
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tic
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tic
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
      desc += "(Trade tick)";
     }
   else
     {
      //--- form an output for an info tick a bit differently
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- return tick description
   return(desc);
  }
```

See also

CustomRatesDelete, CustomRatesUpdate, CustomTicksReplace, CopyTicks, CopyTicksRange

CustomTicksDelete

Deletes all ticks from the price history of the custom symbol in the specified time interval.

```
int  CustomTicksDelete(
   const string      symbol,             // symbol name
   long              from_msc,           // start date
   long              to_msc              // end date
   );
```

Parameters

symbol

[in] Custom symbol name.

from_msc

[in] Time of the first tick in the price history within the specified range to be removed. Time in milliseconds since 01.01.1970.

to_msc

[in] Time of the last tick in the price history within the specified range to be removed. Time in milliseconds since 01.01.1970.

Return Value

Number of deleted ticks or -1 in case of an error.

Example:

```
//+------------------------------------------------------------------+
//|                                           CustomTicksDelete.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"     // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"           // name of the group, in which a sy
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()          // name of a symbol a custom one is

#define   DATATICKS_TO_COPY      UINT_MAX          // number of ticks copied
#define   DATATICKS_TO_DELETE    10                // number of deleted ticks
#define   DATATICKS_TO_PRINT     20                // number of ticks sent to the jour

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBOI

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol has
   if(create!=0 && create!=5304)
      return;

//--- get the standard symbol tick data to the MqlTick array
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- print the time of the first and last received ticks of the standard symbol
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the standard symbol in the journal
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRINT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- add a custom symbol to the MarketWatch window
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- add the tick array data to the custom symbol price history
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", ar
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", add

//--- get the newly added custom symbol tick data to the MqlTick array
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- print the time of the first and last received ticks of the custom symbol
```

```
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the custom symbol in the journal
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- get the tick time in milliseconds, from which we will delete the range of ticks,
   long time_from=array[total-DATATICKS_TO_DELETE-1].time_msc;

//--- delete DATATICKS_TO_DELETE the range of the last ticks of the custom symbol in t
   Print("...");
   start=GetTickCount();
   PrintFormat("Start deleting %u ticks in the history of the custom symbol '%s'", DAT
   int deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, time_from, array[total-2].time_ms
   PrintFormat("Deleted %u ticks in the history of the custom symbol '%s' in %u ms", 

//--- get the newly modified custom symbol tick data to the MqlTick array
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- print the time of the first and last ticks of a custom symbol with a removed tic
   total=(int)array.Size();
   PrintFormat("Time of the first tick from the changed history: %s.%03u, Time of the 
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the custom symbol in the journal
   PrintFormat("\nThe last %d ticks of custom symbol '%s' with modified history:", DAT
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and ex
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully 

         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully

         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Requested 4294967295 ticks to get tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351199027 ti
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 10:10:40.392

   The last 20 ticks for the standard symbol 'EURUSD':
     351199007th Tick: 2024.06.21 10:10:23.045 Bid=1.07032 (Info tick)
     351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
     351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 (Info tick)
     351199010th Tick: 2024.06.21 10:10:25.146 Bid=1.07032 (Info tick)
     351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
     351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
     351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
     351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
     351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
     351199016th Tick: 2024.06.21 10:10:30.258 Bid=1.07036 (Info tick)
     351199017th Tick: 2024.06.21 10:10:30.872 Ask=1.07035 Bid=1.07035 (Info tick)
     351199018th Tick: 2024.06.21 10:10:31.358 Ask=1.07036 (Info tick)
     351199019th Tick: 2024.06.21 10:10:31.859 Ask=1.07037 Bid=1.07037 (Info tick)
     351199020th Tick: 2024.06.21 10:10:32.377 Ask=1.07039 Bid=1.07039 (Info tick)
     351199021th Tick: 2024.06.21 10:10:32.962 Ask=1.0704 Bid=1.0704 (Info tick)
     351199022th Tick: 2024.06.21 10:10:33.961 Ask=1.07039 Bid=1.07039 (Info tick)
     351199023th Tick: 2024.06.21 10:10:34.667 Ask=1.0704 (Info tick)
     351199024th Tick: 2024.06.21 10:10:35.170 Bid=1.0704 (Info tick)
     351199025th Tick: 2024.06.21 10:10:38.266 Ask=1.07041 Bid=1.07041 (Info tick)
     351199026th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
```

```
...
Start of adding 351199027 ticks to the history of the custom symbol 'EURUSD.C'
Added 351199027 ticks to the history of the custom symbol 'EURUSD.C' in 261594 ms
...
Requested 351199027 ticks to get tick history for the symbol 'EURUSD.C'
The tick history for the 'EURUSD.C' symbol is received in the amount of 351199027 t
First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 10:10:40.392

The last 20 ticks for the custom symbol 'EURUSD.C':
  351199007th Tick: 2024.06.21 10:10:23.045 Ask=1.07032 Bid=1.07032 (Info tick)
  351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
  351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 Bid=1.07032 (Info tick)
  351199010th Tick: 2024.06.21 10:10:25.146 Ask=1.07032 Bid=1.07032 (Info tick)
  351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
  351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
  351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
  351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
  351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
  351199016th Tick: 2024.06.21 10:10:30.258 Ask=1.07036 Bid=1.07036 (Info tick)
  351199017th Tick: 2024.06.21 10:10:30.872 Ask=1.07035 Bid=1.07035 (Info tick)
  351199018th Tick: 2024.06.21 10:10:31.358 Ask=1.07036 Bid=1.07036 (Info tick)
  351199019th Tick: 2024.06.21 10:10:31.859 Ask=1.07037 Bid=1.07037 (Info tick)
  351199020th Tick: 2024.06.21 10:10:32.377 Ask=1.07039 Bid=1.07039 (Info tick)
  351199021th Tick: 2024.06.21 10:10:32.962 Ask=1.0704 Bid=1.0704 (Info tick)
  351199022th Tick: 2024.06.21 10:10:33.961 Ask=1.07039 Bid=1.07039 (Info tick)
  351199023th Tick: 2024.06.21 10:10:34.667 Ask=1.0704 Bid=1.0704 (Info tick)
  351199024th Tick: 2024.06.21 10:10:35.170 Ask=1.0704 Bid=1.0704 (Info tick)
  351199025th Tick: 2024.06.21 10:10:38.266 Ask=1.07041 Bid=1.07041 (Info tick)
  351199026th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
...
Start deleting 10 ticks in the history of the custom symbol 'EURUSD.C'
Deleted 10 ticks in the history of the custom symbol 'EURUSD.C' in 188 ms
...
Requested 351199027 ticks to get tick history for the symbol 'EURUSD.C'
The tick history for the 'EURUSD.C' symbol is received in the amount of 351199017 t
Time of the first tick from the changed history: 2011.12.19 00:00:08.000, Time of t

The last 20 ticks of custom symbol 'EURUSD.C' with modified history:
  351198997th Tick: 2024.06.21 10:10:14.935 Ask=1.07036 Bid=1.07036 (Info tick)
  351198998th Tick: 2024.06.21 10:10:15.533 Ask=1.07035 Bid=1.07035 (Info tick)
  351198999th Tick: 2024.06.21 10:10:17.736 Ask=1.07036 Bid=1.07036 (Info tick)
  351199000th Tick: 2024.06.21 10:10:18.540 Ask=1.07037 Bid=1.07037 (Info tick)
  351199001th Tick: 2024.06.21 10:10:19.046 Ask=1.07038 Bid=1.07038 (Info tick)
  351199002th Tick: 2024.06.21 10:10:19.542 Ask=1.07036 Bid=1.07036 (Info tick)
  351199003th Tick: 2024.06.21 10:10:20.041 Ask=1.07035 Bid=1.07035 (Info tick)
  351199004th Tick: 2024.06.21 10:10:21.041 Ask=1.07035 Bid=1.07035 (Info tick)
  351199005th Tick: 2024.06.21 10:10:21.544 Ask=1.07032 Bid=1.07032 (Info tick)
  351199006th Tick: 2024.06.21 10:10:22.344 Ask=1.07032 Bid=1.07032 (Info tick)
  351199007th Tick: 2024.06.21 10:10:23.045 Ask=1.07032 Bid=1.07032 (Info tick)
```

```
     351199008th Tick: 2024.06.21 10:10:24.045 Ask=1.07031 Bid=1.07031 (Info tick)
     351199009th Tick: 2024.06.21 10:10:24.545 Ask=1.07032 Bid=1.07032 (Info tick)
     351199010th Tick: 2024.06.21 10:10:25.146 Ask=1.07032 Bid=1.07032 (Info tick)
     351199011th Tick: 2024.06.21 10:10:25.649 Ask=1.07037 Bid=1.07037 (Info tick)
     351199012th Tick: 2024.06.21 10:10:27.050 Ask=1.07036 Bid=1.07036 (Info tick)
     351199013th Tick: 2024.06.21 10:10:28.153 Ask=1.07039 Bid=1.07039 (Info tick)
     351199014th Tick: 2024.06.21 10:10:29.157 Ask=1.07037 Bid=1.07037 (Info tick)
     351199015th Tick: 2024.06.21 10:10:29.658 Ask=1.07036 Bid=1.07036 (Info tick)
     351199016th Tick: 2024.06.21 10:10:40.392 Ask=1.07042 Bid=1.07042 (Info tick)
   */
  }
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const stri
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name,
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Get the specified number of ticks in the array                   |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- notify of the start of loading historical data
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, s

//--- make 3 attempts to receive ticks
   int attempts=0;
   while(attempts<3)
     {
      //--- measure the start time before receiving the ticks
      uint start=GetTickCount();

      //--- request the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
      int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
      if(received!=-1)
        {
         //--- display information about the number of ticks and spent time
         PrintFormat("The tick history for the '%s' symbol is received in the amount 

         //--- if the tick history is synchronized, the error code is equal to zero -
         if(GetLastError()==0)
            return(true);

         PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms
                     symbol, received, GetTickCount()-start, GetLastError());
        }
      //--- count attempts
      attempts++;
      //--- a one-second pause to wait for the end of synchronization of the tick dat
      Sleep(1000);
     }
//--- failed to copy ticks in 3 attempts
   return(false);
  }
//+------------------------------------------------------------------+
//| return the string description of a tick                          |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTE
//--- check tick flags
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)   == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)  == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)   == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)   == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)  == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- check the tick for trading flags first (there are none for CustomTicksAdd())
   if(buy_tick || sell_tick)
     {
      //--- form an output for a trading tick
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tic
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tic
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
      desc += "(Trade tick)";
     }
   else
     {
      //--- form an output for an info tick a bit differently
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- return tick description
   return(desc);
  }
```

See also

CustomRatesDelete, CustomRatesUpdate, CustomTicksReplace, CopyTicks, CopyTicksRange

CustomTicksReplace

Fully replaces the price history of the custom symbol within the specified time interval with the data from the MqlTick type array.

```
int  CustomTicksReplace(
   const string      symbol,            // symbol name
   long              from_msc,          // start date
   long              to_msc,            // end date
   const MqlTick&    ticks[],           // array for the data to be applied to a custo
   uint              count=WHOLE_ARRAY  // number of the ticks[] array elements to be
   );
```

Parameters

symbol

[in] Custom symbol name.

from_msc

[in] Time of the first tick in the price history within the specified range to be removed. Time in milliseconds since 01.01.1970.

to_msc

[in] Time of the last tick in the price history within the specified range to be removed. Time in milliseconds since 01.01.1970.

ticks[]

[in] Array of the MqlTick type tick data ordered in time in ascending order.

count=WHOLE_ARRAY

[in] Number of the *ticks[]* array elements to be used for replacement in the specified time interval. WHOLE_ARRAY means that all *ticks[]* array elements should be used.

Return Value

Number of updated ticks or -1 in case of an error.

Note

Since several ticks may often have the same time up to a millisecond in a stream of quotes (accurate tick time is stored in the *time_msc* field of the MqlTick structure), the CustomTicksReplace function does not automatically sort out the *ticks[]* array elements by time. Therefore, the array of ticks must be pre-arranged in time ascending order.

The ticks are replaced consecutively, day after day, until the time specified in *to_msc* or until an error occurs. The first day from the specified range is processed followed by the next one, etc. As soon as the mismatch between the tick time and the ascending (non-descending) order is detected, the tick replacement stops on the current day. All ticks from the previous days are successfully replaced, while the current day (at the moment of a wrong tick) and all the remaining days in the specified interval remain unchanged.

If the *ticks[]* array contains no tick data for any day (generally, any time interval), a "hole" corresponding to the missing data appears in the custom symbol history after the tick data from `ticks[]` are applied. In other words, the call of CustomTicksReplace with missing ticks is equivalent to deleting part of the tick history, as if CustomTicksDelete with the "hole" interval is called.

If the tick database has no data for the specified time interval, CustomTicksReplace will add to the database ticks form the ticks[] array.

The CustomTicksReplace function works directly with the tick database.

Example:

```
//+------------------------------------------------------------------+
//|                                           CustomTicksReplace.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#define   CUSTOM_SYMBOL_NAME     Symbol()+".C"      // custom symbol name
#define   CUSTOM_SYMBOL_PATH     "Forex"            // name of the group, in which a sy
#define   CUSTOM_SYMBOL_ORIGIN   Symbol()           // name of a symbol a custom one i

#define   DATATICKS_TO_COPY      UINT_MAX           // number of ticks copied
#define   DATATICKS_TO_PRINT     20                 // number of ticks sent to the jou

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the error code when creating a custom symbol
   int create=CreateCustomSymbol(CUSTOM_SYMBOL_NAME, CUSTOM_SYMBOL_PATH, CUSTOM_SYMBO

//--- if the error code is not 0 (successful symbol creation) and not 5304 (symbol ha
   if(create!=0 && create!=5304)
      return;

//--- get the standard symbol tick data to the MqlTick array
   MqlTick array[]={};
   if(!GetTicksToArray(CUSTOM_SYMBOL_ORIGIN, DATATICKS_TO_COPY, array))
      return;

//--- print the time of the first and last received ticks of the standard symbol
   int total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),
//--- print DATATICKS_TO_PRINT last ticks of the standard symbol in the journal
   PrintFormat("\nThe last %d ticks for the standard symbol '%s':", DATATICKS_TO_PRIN
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- add a custom symbol to the MarketWatch window
   ResetLastError();
   if(!SymbolSelect(CUSTOM_SYMBOL_NAME, true))
     {
      Print("SymbolSelect() failed. Error ", GetLastError());
      return;
     }

//--- add the tick array data to the custom symbol price history
   Print("...");
   uint start=GetTickCount();
   PrintFormat("Start of adding %u ticks to the history of the custom symbol '%s'", a
   int added=CustomTicksAdd(CUSTOM_SYMBOL_NAME, array);
   PrintFormat("Added %u ticks to the history of the custom symbol '%s' in %u ms", ad

//--- get the newly added custom symbol tick data to the MqlTick array
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- print the time of the first and last received ticks of the custom symbol
   total=(int)array.Size();
   PrintFormat("First tick time: %s.%03u, Last tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), arra
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last ticks of the custom symbol in the journal
   PrintFormat("\nThe last %d ticks for the custom symbol '%s':", DATATICKS_TO_PRINT,
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Tick: %s", i, GetTickDescription(array[i]));
     }

//--- now change Ask and Bid tick values in the array using the equation Ask(Symbol)
   for(int i=0; i<total; i++)
     {
      array[i].ask = (array[i].ask !=0 ? 1.0 / array[i].ask : array[i].ask);
      array[i].bid = (array[i].bid !=0 ? 1.0 / array[i].bid : array[i].bid);
     }
   Print("\nNow the ticks are changed");

//--- replace the tick history of the custom symbol with data from the modified array
   Print("...");
   start=GetTickCount();
   PrintFormat("Start replacing %u changed ticks in the history of the custom symbol
   int replaced=CustomTicksReplace(CUSTOM_SYMBOL_NAME, array[0].time_msc, array[total-
   PrintFormat("Replaced %u ticks in the history of the custom symbol '%s' in %u ms",

//--- get the newly replaced custom symbol tick data to the MqlTick array
   Print("...");
   if(!GetTicksToArray(CUSTOM_SYMBOL_NAME, array.Size(), array))
      return;

//--- print the time of the first and last received modified ticks of the custom symbo
   total=(int)array.Size();
   PrintFormat("First changed tick time: %s.%03u, Last changed tick time: %s.%03u",
               TimeToString(array[0].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS), array
               TimeToString(array[total-1].time, TIME_DATE|TIME_MINUTES|TIME_SECONDS),

//--- print DATATICKS_TO_PRINT last modified ticks of the custom symbol in the journal
   PrintFormat("\nThe last %d changed ticks for the custom symbol '%s':", DATATICKS_T
   for(int i=total-DATATICKS_TO_PRINT; i<total; i++)
     {
      if(i<0)
         continue;
      PrintFormat("  %dth Changed tick: %s", i, GetTickDescription(array[i]));
     }

//--- display a hint about the script termination keys on the chart comment
   Comment(StringFormat("Press 'Esc' to exit or 'Del' to delete the '%s' symbol and e
//--- wait for pressing the Esc or Del keys to exit in an endless loop
   while(!IsStopped() && TerminalInfoInteger(TERMINAL_KEYSTATE_ESCAPE)==0)
     {
      Sleep(16);
      //--- when pressing Del, delete the created custom symbol and its data
      if(TerminalInfoInteger(TERMINAL_KEYSTATE_DELETE)<0)
        {
         //--- delete bar data
         int deleted=CustomRatesDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history bars of the custom symbol '%s' were successfully

         //--- delete tick data
         deleted=CustomTicksDelete(CUSTOM_SYMBOL_NAME, 0, LONG_MAX);
         if(deleted>0)
            PrintFormat("%d history ticks of the custom symbol '%s' were successfully
         //--- delete symbol
         if(DeleteCustomSymbol(CUSTOM_SYMBOL_NAME))
            PrintFormat("Custom symbol '%s' deleted successfully", CUSTOM_SYMBOL_NAME)
         break;
        }
     }
//--- clear the chart before exiting
   Comment("");
   /*
   result:
   Requested 4294967295 ticks to get tick history for the symbol 'EURUSD'
   The tick history for the 'EURUSD' symbol is received in the amount of 351195822 tic
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 08:39:03.113

   The last 20 ticks for the standard symbol 'EURUSD':
     351195802th Tick: 2024.06.21 08:38:10.076 Ask=1.07194 (Info tick)
     351195803th Tick: 2024.06.21 08:38:13.162 Ask=1.07195 (Info tick)
     351195804th Tick: 2024.06.21 08:38:13.872 Bid=1.07195 (Info tick)
     351195805th Tick: 2024.06.21 08:38:14.866 Ask=1.07194 Bid=1.07194 (Info tick)
     351195806th Tick: 2024.06.21 08:38:17.374 Bid=1.07194 (Info tick)
     351195807th Tick: 2024.06.21 08:38:18.883 Bid=1.07194 (Info tick)
     351195808th Tick: 2024.06.21 08:38:19.771 Bid=1.07194 (Info tick)
     351195809th Tick: 2024.06.21 08:38:20.873 Ask=1.07195 Bid=1.07195 (Info tick)
     351195810th Tick: 2024.06.21 08:38:22.278 Ask=1.07196 Bid=1.07196 (Info tick)
     351195811th Tick: 2024.06.21 08:38:22.775 Bid=1.07196 (Info tick)
     351195812th Tick: 2024.06.21 08:38:23.477 Bid=1.07196 (Info tick)
     351195813th Tick: 2024.06.21 08:38:38.194 Ask=1.07197 (Info tick)
     351195814th Tick: 2024.06.21 08:38:38.789 Ask=1.07196 (Info tick)
     351195815th Tick: 2024.06.21 08:38:39.290 Ask=1.07197 (Info tick)
     351195816th Tick: 2024.06.21 08:38:43.695 Ask=1.07196 (Info tick)
     351195817th Tick: 2024.06.21 08:38:52.203 Ask=1.07195 Bid=1.07195 (Info tick)
     351195818th Tick: 2024.06.21 08:38:55.105 Ask=1.07196 Bid=1.07196 (Info tick)
     351195819th Tick: 2024.06.21 08:38:57.607 Ask=1.07195 Bid=1.07195 (Info tick)
     351195820th Tick: 2024.06.21 08:39:00.512 Ask=1.07196 Bid=1.07196 (Info tick)
     351195821th Tick: 2024.06.21 08:39:03.113 Ask=1.07195 Bid=1.07195 (Info tick)
   ...
   Start of adding 351195822 ticks to the history of the custom symbol 'EURUSD.C'
   Added 351195822 ticks to the history of the custom symbol 'EURUSD.C' in 349407 ms
   ...
   Requested 351195822 ticks to get tick history for the symbol 'EURUSD.C'
   The tick history for the 'EURUSD.C' symbol is received in the amount of 351195822 t
   First tick time: 2011.12.19 00:00:08.000, Last tick time: 2024.06.21 08:39:03.113

   The last 20 ticks for the custom symbol 'EURUSD.C':
     351195802th Tick: 2024.06.21 08:38:10.076 Ask=1.07194 Bid=1.07194 (Info tick)
     351195803th Tick: 2024.06.21 08:38:13.162 Ask=1.07195 Bid=1.07195 (Info tick)
     351195804th Tick: 2024.06.21 08:38:13.872 Ask=1.07195 Bid=1.07195 (Info tick)
     351195805th Tick: 2024.06.21 08:38:14.866 Ask=1.07194 Bid=1.07194 (Info tick)
   351195806th Tick: 2024.06.21 08:38:17.374 Ask=1.07194 Bid=1.07194 (Info tick)
   351195807th Tick: 2024.06.21 08:38:18.883 Ask=1.07194 Bid=1.07194 (Info tick)
   351195808th Tick: 2024.06.21 08:38:19.771 Ask=1.07194 Bid=1.07194 (Info tick)
   351195809th Tick: 2024.06.21 08:38:20.873 Ask=1.07195 Bid=1.07195 (Info tick)
   351195810th Tick: 2024.06.21 08:38:22.278 Ask=1.07196 Bid=1.07196 (Info tick)
   351195811th Tick: 2024.06.21 08:38:22.775 Ask=1.07196 Bid=1.07196 (Info tick)
   351195812th Tick: 2024.06.21 08:38:23.477 Ask=1.07196 Bid=1.07196 (Info tick)
   351195813th Tick: 2024.06.21 08:38:38.194 Ask=1.07197 Bid=1.07197 (Info tick)
   351195814th Tick: 2024.06.21 08:38:38.789 Ask=1.07196 Bid=1.07196 (Info tick)
   351195815th Tick: 2024.06.21 08:38:39.290 Ask=1.07197 Bid=1.07197 (Info tick)
   351195816th Tick: 2024.06.21 08:38:43.695 Ask=1.07196 Bid=1.07196 (Info tick)
   351195817th Tick: 2024.06.21 08:38:52.203 Ask=1.07195 Bid=1.07195 (Info tick)
   351195818th Tick: 2024.06.21 08:38:55.105 Ask=1.07196 Bid=1.07196 (Info tick)
   351195819th Tick: 2024.06.21 08:38:57.607 Ask=1.07195 Bid=1.07195 (Info tick)
   351195820th Tick: 2024.06.21 08:39:00.512 Ask=1.07196 Bid=1.07196 (Info tick)
   351195821th Tick: 2024.06.21 08:39:03.113 Ask=1.07195 Bid=1.07195 (Info tick)

 Now the ticks are changed
 ...
 Start replacing 351195822 changed ticks in the history of the custom symbol 'EURUS
 Replaced 351195822 ticks in the history of the custom symbol 'EURUSD.C' in 452266
 ...
 Requested 351195822 ticks to get tick history for the symbol 'EURUSD.C'
 The tick history for the 'EURUSD.C' symbol is received in the amount of 351195822
 First changed tick time: 2011.12.19 00:00:08.000, Last changed tick time: 2024.06.

 The last 20 changed ticks for the custom symbol 'EURUSD.C':
   351195802th Changed tick: 2024.06.21 08:38:10.076 Ask=0.93289 Bid=0.93289 (Info
   351195803th Changed tick: 2024.06.21 08:38:13.162 Ask=0.93288 Bid=0.93288 (Info
   351195804th Changed tick: 2024.06.21 08:38:13.872 Ask=0.93288 Bid=0.93288 (Info
   351195805th Changed tick: 2024.06.21 08:38:14.866 Ask=0.93289 Bid=0.93289 (Info
   351195806th Changed tick: 2024.06.21 08:38:17.374 Ask=0.93289 Bid=0.93289 (Info
   351195807th Changed tick: 2024.06.21 08:38:18.883 Ask=0.93289 Bid=0.93289 (Info
   351195808th Changed tick: 2024.06.21 08:38:19.771 Ask=0.93289 Bid=0.93289 (Info
   351195809th Changed tick: 2024.06.21 08:38:20.873 Ask=0.93288 Bid=0.93288 (Info
   351195810th Changed tick: 2024.06.21 08:38:22.278 Ask=0.93287 Bid=0.93287 (Info
   351195811th Changed tick: 2024.06.21 08:38:22.775 Ask=0.93287 Bid=0.93287 (Info
   351195812th Changed tick: 2024.06.21 08:38:23.477 Ask=0.93287 Bid=0.93287 (Info
   351195813th Changed tick: 2024.06.21 08:38:38.194 Ask=0.93286 Bid=0.93286 (Info
   351195814th Changed tick: 2024.06.21 08:38:38.789 Ask=0.93287 Bid=0.93287 (Info
   351195815th Changed tick: 2024.06.21 08:38:39.290 Ask=0.93286 Bid=0.93286 (Info
   351195816th Changed tick: 2024.06.21 08:38:43.695 Ask=0.93287 Bid=0.93287 (Info
   351195817th Changed tick: 2024.06.21 08:38:52.203 Ask=0.93288 Bid=0.93288 (Info
   351195818th Changed tick: 2024.06.21 08:38:55.105 Ask=0.93287 Bid=0.93287 (Info
   351195819th Changed tick: 2024.06.21 08:38:57.607 Ask=0.93288 Bid=0.93288 (Info
   351195820th Changed tick: 2024.06.21 08:39:00.512 Ask=0.93287 Bid=0.93287 (Info
   351195821th Changed tick: 2024.06.21 08:39:03.113 Ask=0.93288 Bid=0.93288 (Info
 */
}
```

```
//+------------------------------------------------------------------+
//| Create a custom symbol, return an error code                     |
//+------------------------------------------------------------------+
int CreateCustomSymbol(const string symbol_name, const string symbol_path, const strin
  {
//--- define the name of a symbol a custom one is to be based on
   string origin=(symbol_origin==NULL ? Symbol() : symbol_origin);

//--- if failed to create a custom symbol and this is not error 5304, report this in t
   ResetLastError();
   int error=0;
   if(!CustomSymbolCreate(symbol_name, symbol_path, origin))
     {
      error=GetLastError();
      if(error!=5304)
         PrintFormat("CustomSymbolCreate(%s, %s, %s) failed. Error %d", symbol_name, s
     }
//--- successful
   return(error);
  }
//+------------------------------------------------------------------+
//| Remove a custom symbol                                           |
//+------------------------------------------------------------------+
bool DeleteCustomSymbol(const string symbol_name)
  {
//--- hide the symbol from the Market Watch window
   ResetLastError();
   if(!SymbolSelect(symbol_name, false))
     {
      PrintFormat("SymbolSelect(%s, false) failed. Error %d", GetLastError());
      return(false);
     }

//--- if failed to delete a custom symbol, report this in the journal and return 'fals
   ResetLastError();
   if(!CustomSymbolDelete(symbol_name))
     {
      PrintFormat("CustomSymbolDelete(%s) failed. Error %d", symbol_name, GetLastErro
      return(false);
     }
//--- successful
   return(true);
  }
//+------------------------------------------------------------------+
//| Get the specified number of ticks in the array                   |
//+------------------------------------------------------------------+
bool GetTicksToArray(const string symbol, const uint count, MqlTick &array[])
  {
//--- notify of the start of loading historical data
```

```
   PrintFormat("Requested %u ticks to get tick history for the symbol '%s'", count, sy

//--- make 3 attempts to receive ticks
   int attempts=0;
   while(attempts<3)
     {
      //--- measure the start time before receiving the ticks
      uint start=GetTickCount();

      //--- request the tick history since 1970.01.01 00:00.001 (parameter from=1 ms)
      int received=CopyTicks(symbol, array, COPY_TICKS_ALL, 1, count);
      if(received!=-1)
        {
         //--- display information about the number of ticks and spent time
         PrintFormat("The tick history for the '%s' symbol is received in the amount

         //--- if the tick history is synchronized, the error code is equal to zero -
         if(GetLastError()==0)
            return(true);

         PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms.
                     symbol, received, GetTickCount()-start, GetLastError());
        }
      //--- count attempts
      attempts++;
      //--- a one-second pause to wait for the end of synchronization of the tick data
      Sleep(1000);
     }
//--- failed to copy ticks in 3 attempts
   return(false);
  }
//+------------------------------------------------------------------+
//| return the string description of a tick                          |
//+------------------------------------------------------------------+
string GetTickDescription(MqlTick &tick)
  {
   string desc=StringFormat("%s.%03u ", TimeToString(tick.time, TIME_DATE|TIME_MINUTES

//--- check tick flags
   bool buy_tick   = ((tick.flags &TICK_FLAG_BUY)   == TICK_FLAG_BUY);
   bool sell_tick  = ((tick.flags &TICK_FLAG_SELL)  == TICK_FLAG_SELL);
   bool ask_tick   = ((tick.flags &TICK_FLAG_ASK)   == TICK_FLAG_ASK);
   bool bid_tick   = ((tick.flags &TICK_FLAG_BID)   == TICK_FLAG_BID);
   bool last_tick  = ((tick.flags &TICK_FLAG_LAST)  == TICK_FLAG_LAST);
   bool volume_tick= ((tick.flags &TICK_FLAG_VOLUME)== TICK_FLAG_VOLUME);

//--- check the tick for trading flags first (there are none for CustomTicksAdd())
   if(buy_tick || sell_tick)
     {
      //--- form an output for a trading tick
      desc += (buy_tick ? StringFormat("Buy Tick: Last=%G Volume=%d ", tick.last, tick
      desc += (sell_tick? StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last, tick
      desc += (ask_tick ? StringFormat("Ask=%G ", tick.ask) : "");
      desc += (bid_tick ? StringFormat("Bid=%G ", tick.ask) : "");
      desc += "(Trade tick)";
     }
   else
     {
      //--- form an output for an info tick a bit differently
      desc += (ask_tick   ? StringFormat("Ask=%G ",  tick.ask)    : "");
      desc += (bid_tick   ? StringFormat("Bid=%G ",  tick.ask)    : "");
      desc += (last_tick  ? StringFormat("Last=%G ", tick.last)   : "");
      desc += (volume_tick? StringFormat("Volume=%d ",tick.volume): "");
      desc += "(Info tick)";
     }
//--- return tick description
   return(desc);
  }
```

See also

CustomRatesDelete, CustomRatesUpdate, CustomTicksDelete, CopyTicks, CopyTicksRange

CustomBookAdd

Passes the status of the Depth of Market for a custom symbol. The function allows broadcasting the Depth of Market as if the prices arrive from a broker's server.

```
bool  CustomBookAdd(
   const string          symbol,            // symbol name
   const MqlBookInfo&    books[]            // array with descriptions of the Depth of M
   uint                  count=WHOLE_ARRAY  // number of elements to be used
   );
```

Parameters

symbol

[in] Custom symbol name.

books[]

[in] The array of MqlBookInfo type data fully describing the Depth of Market status — all buy and sell requests. The passed Depth of Market status completely replaces the previous one.

count=WHOLE_ARRAY

[in] The number of 'books' array elements to be passed to the function. The entire array is used by default.

Return Value

true - success, otherwise - false. To get information about the error, call the GetLastError() function.

Note

The CustomBookAdd function works only for custom symbols the Depth of Market is opened for — via the platform interface or the MarketBookAdd function.

When throwing the Depth of Market in, the symbol's Bid and Ask prices are not updated. You should control the change of the best prices and throw in the ticks using CustomTicksAdd.

The function verifies the accuracy of transmitted data: the type, price and volume must be indicated for each element. Furthermore, MqlBookInfo.volume and MqlBookInfo.volume_real must not be zero or negative; if both volumes are negative, this will be considered an error. You can specify any of the volumes or both of them: the one that is indicated or is positive will be used:

```
volume=-1 && volume_real=2 — volume_real=2 will be used,
a volume=3 && volume_real=0 — volume=3 will be used.
```

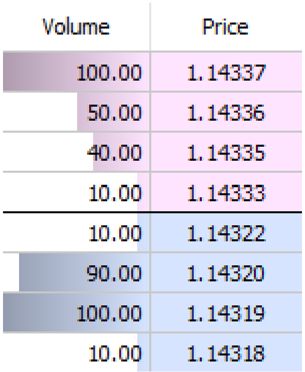
The MqlBookInfo.volume_real extended accuracy volume has a higher priority over the regular MqlBookInfo.volume. If both values are specified for the Depth of Market element, the volume_real one is used.

The order of the MqlBookInfo elements in the 'books' array does not matter. When saving the data, the terminal sorts them by price on its own.

When saving data, the "Book depth" (SYMBOL_TICKS_BOOKDEPTH) parameter of the recipient custom symbol is checked. If the number of sell requests exceeds this value in the passed Depth of Market, the excess levels are discarded. The same is true for buy requests.

Sample filling of the 'books' array:

Depth of Market status	Filling books[]


Volume Price
100.00 1.14337
50.00 1.14336
40.00 1.14335
10.00 1.14333
10.00 1.14322
90.00 1.14320
100.00 1.14319
10.00 1.14318
	books[0].type=BOOK_TYPE_SELL;
	books[0].price=1.14337;
	books[0].volume=100;
	books[1].type=BOOK_TYPE_SELL;
	books[1].price=1.14330;
	books[1].volume=50;
	books[2].type=BOOK_TYPE_SELL;
	books[2].price=1.14335;
	books[2].volume=40;
	books[3].type=BOOK_TYPE_SELL;
	books[3].price=1.14333;
	books[3].volume=10;
	books[4].type=BOOK_TYPE_BUY;
	books[4].price=1.14322;
	books[4].volume=10;
	books[5].type=BOOK_TYPE_BUY;
	books[5].price=1.14320;
	books[5].volume=90;
	books[6].type=BOOK_TYPE_BUY;
	books[6].price=1.14319;
	books[6].volume=100;
	books[7].type=BOOK_TYPE_BUY;
	books[7].price=1.14318;
	books[7].volume=10;

Example:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- enable the Depth of Market for a symbol we are to retrieve data from
   MarketBookAdd(Symbol());
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
  }
//+------------------------------------------------------------------+
//| Tick function                                                    |
//+------------------------------------------------------------------+
void OnTick(void)
  {
   MqlTick ticks[];
   ArrayResize(ticks,1);
//--- copy the current prices from the common symbol to the custom one
   if(SymbolInfoTick(Symbol(),ticks[0]))
     {
      string symbol_name=Symbol()+".SYN";
      CustomTicksAdd(symbol_name,ticks);
     }
  }
//+------------------------------------------------------------------+
//| Book function                                                    |
//+------------------------------------------------------------------+
void OnBookEvent(const string &book_symbol)
  {
//--- copy the current Depth of Market status from the common symbol to the custom one
   if(book_symbol==Symbol())
     {
      MqlBookInfo book_array[];
      if(MarketBookGet(Symbol(),book_array))
        {
         string symbol_name=Symbol()+".SYN";
         CustomBookAdd(symbol_name,book_array);
        }
     }
  }
//+------------------------------------------------------------------+
```

See also

MarketBookAdd, CustomTicksAdd, OnBookEvent

Chart Operations

Functions for setting chart properties (ChartSetInteger, ChartSetDouble, ChartSetString) are asynchronous and are used for sending update commands to a chart. If these functions are executed successfully, the command is included in the common queue of the chart events. Chart property changes are implemented along with handling of the events queue of this chart.

Thus, do not expect an immediate update of the chart after calling asynchronous functions. Use the ChartRedraw() function to forcedly update the chart appearance and properties.

Function	Action
ChartApplyTemplate	Applies a specific template from a specified file to the chart
ChartSaveTemplate	Saves current chart settings in a template with a specified name
ChartWindowFind	Returns the number of a subwindow where an indicator is drawn
ChartTimePriceToXY	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates
ChartXYToTimePrice	Converts the X and Y coordinates on a chart to the time and price values
ChartOpen	Opens a new chart with the specified symbol and period
ChartClose	Closes the specified chart
ChartFirst	Returns the ID of the first chart of the client terminal
ChartNext	Returns the chart ID of the chart next to the specified one
ChartSymbol	Returns the symbol name of the specified chart
ChartPeriod	Returns the period value of the specified chart
ChartRedraw	Calls a forced redrawing of a specified chart
ChartSetDouble	Sets the double value for a corresponding property of the specified chart
ChartSetInteger	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart
ChartSetString	Sets the string value for a corresponding property of the specified chart
ChartGetDouble	Returns the double value property of the specified chart
ChartGetInteger	Returns the integer value property of the specified chart
ChartGetString	Returns the string value property of the specified chart
ChartNavigate	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart
ChartID	Returns the ID of the current chart

Function	Action
ChartIndicatorAdd	Adds an indicator with the specified handle into a specified chart window
ChartIndicatorDelete	Removes an indicator with a specified name from the specified chart window
ChartIndicatorGet	Returns the handle of the indicator with the specified short name in the specified chart window
ChartIndicatorName	Returns the short name of the indicator by the number in the indicators list on the specified chart window
ChartIndicatorsTotal	Returns the number of all indicators applied to the specified chart window.
ChartWindowOnDropped	Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to
ChartPriceOnDropped	Returns the price coordinate of the chart point the Expert Advisor or script has been dropped to
ChartTimeOnDropped	Returns the time coordinate of the chart point the Expert Advisor or script has been dropped to
ChartXOnDropped	Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to
ChartYOnDropped	Returns the Y coordinate of the chart point the Expert Advisor or script has been dropped to
ChartSetSymbolPeriod	Changes the symbol value and a period of the specified chart
ChartScreenShot	Provides a screenshot of the chart of its current state in a GIF, PNG or BMP format depending on specified extension

ChartApplyTemplate

Applies a specific template from a specified file to the chart. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool  ChartApplyTemplate(
   long           chart_id,      // Chart ID
   const string   filename       // Template file name
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The name of the file containing the template.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the error, call the GetLastError() function.

Note

The Expert Advisor will be unloaded and will not be able to continue operating in case of successful loading of a new template to the chart it is attached to.

When applying the template to the chart, trade permissions may be limited due to security reasons:

> **Live trading permission cannot be extended for the Expert Advisors launched by applying the template using ChartApplyTemplate() function.**

If the mql5-program calling ChartApplyTemplate() function has no permission to trade, the Expert Advisor launched via the template will also not be able to trade regardless of the template settings.

If the mql5-program calling ChartApplyTemplate() function has permission to trade, while there is no such permission in the template settings, the Expert Advisor launched via the template will not be able to trade.

Using Templates

The resources of the MQL5 language allow setting multiple chart properties, including colors using the ChartSetInteger() function:

- Chart background color;
- Color of the axes, scale and the OHLC line;
- Grid color;
- Color of volumes and position open levels;
- Color of the up bar, shadow and edge of a bullish candlestick;
- Color of the down bar, shadow and edge of a bearish candlestick;
- Color of the chart line and Doji candlesticks;

- Color of the bullish candlestick body;
- Color of the bearish candlestick body;
- Color of the Bid price line;
- Color of the Ask price line;
- Color of the line of the last deal price (Last);
- Color of the stop order levels (Stop Loss and Take Profit).

Besides, there can be multiple graphical objects and indicators on a chart. You may set up a chart with all the necessary indicators once and then save it as a template. Such a template can be applied to any chart.

The ChartApplyTemplate() function is intended for using a previously saved template, and it can be used in any mql5 program. The path to the file that stores the template is passed as the second parameter to ChartApplyTemplate(). The template file is searched according to the following rules:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, the template is searched for relative to the path _terminal_data_directory\MQL5,
- if there is no backslash, the template is searched for relative to the executable EX5 file, in which ChartApplyTemplate() is called;
- if a template is not found in the first two variants, the search is performed in the folder terminal_directory\Profiles\Templates\.

Here terminal_directory is the folder from which the MetaTrader 5 Client Terminal is running, and terminal_data_directory is the folder, in which editable files are stored, its location depends on the operating system, user name and computer's security settings. Normally they are different folders, but in some cases they may coincide.

The location of folders terminal_data_directory and terminal_directory can be obtained using the TerminalInfoString() function.

```
//--- directory from which the terminal is started
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
   Print("Terminal directory:",terminal_path);
//--- terminal data directory, in which the MQL5 folder with EAs and indicators is lo
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   Print("Terminal data directory:",terminal_data_path);
```

For example:

```
//--- search for a template in terminal_data_directory\MQL5\
ChartApplyTemplate(0,"\\first_template.tpl"))
//--- search for a template in directory_of_EX5_file\, then in folder terminal_data_d
ChartApplyTemplate(0,"second_template.tpl"))
//--- search for a template in directory_of_EX5_file\My_templates\, then in folder te
ChartApplyTemplate(0,"My_templates\\third_template.tpl"))
```

Templates are not resources, they cannot be included into an executable EX5 file.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- example of applying template, located in \MQL5\Files
   if(FileIsExist("my_template.tpl"))
     {
      Print("The file my_template.tpl found in \Files'");
      //--- apply template
      if(ChartApplyTemplate(0,"\\Files\\my_template.tpl"))
        {
         Print("The template 'my_template.tpl' applied successfully");
         //--- redraw chart
         ChartRedraw();
        }
      else
         Print("Failed to apply 'my_template.tpl', error code ",GetLastError());
     }
   else
     {
      Print("File 'my_template.tpl' not found in "
            +TerminalInfoString(TERMINAL_PATH)+"\\MQL5\\Files");
     }
  }
```

See also

[Resources]

ChartSaveTemplate

Saves current chart settings in a template with a specified name.

```
bool  ChartSaveTemplate(
   long            chart_id,      // Chart ID
   const string  filename       // Filename to save the template
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The filename to save the template. The ".tpl" extension will be added to the filename automatically; there is no need to specify it. The template is saved in **data_folder**\Profiles\Templates\ and can be used for manual application in the terminal. If a template with the same filename already exists, the contents of this file will be overwritten.

Return Value

If successful, the function returns true, otherwise it returns false. To get information about the error, call the GetLastError() function.

Note

Using templates, you can save chart settings with all applied indicators and graphical objects, to then apply it to another chart.

Example:

```
//+------------------------------------------------------------------+
//|                                       Test_ChartSaveTemplate.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input string               symbol="GBPUSD";  // The symbol of a new chart
input ENUM_TIMEFRAMES      period=PERIOD_H3; // The timeframe of a new chart
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- First attach indicators to the chart
   int handle;
//--- Prepare the indicator for use
```

```
   if(!PrepareZigzag(NULL,0,handle)) return; // Failed, so exit
//--- Attach the indicator to the current chart, but in a separate window.
   if(!ChartIndicatorAdd(0,1,handle))
     {
      PrintFormat("Failed to attach to chart %s/%s an indicator with the handle=%d. E
                  _Symbol,
                  EnumToString(_Period),
                  handle,
                  GetLastError());
      //--- Terminate the program operation
      return;
     }
//--- Refresh the chart to see the indicator
   ChartRedraw();
//--- Find the last two last fractures of the zigzag
   double two_values[];
   datetime two_times[];
   if(!GetLastTwoFractures(two_values,two_times,handle))
     {
      PrintFormat("Failed to find two last fractures in the Zigzag!");
      //--- Terminate the program operation
      return;
     }
//--- Now attach a standard deviation channel
   string channel="StdDeviation Channel";
   if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
     {
      PrintFormat("Failed to create object %s. Error code %d",
                  EnumToString(OBJ_STDDEVCHANNEL),GetLastError());
      return;
     }
   else
     {
      //--- The channel has been created, define the second point
      ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
      //--- Set a tooltip text for the channel
      ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL5 Help");
      //--- Refresh the chart
      ChartRedraw();
     }
//--- Save the result in a template
   ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- Open a new chart and apply a saved template to it
   long new_chart=ChartOpen(symbol,period);
   //--- Enable tooltips for graphical objects
   ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
   if(new_chart!=0)
     {
      //--- Apply the saved template to a chart
      ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
     }
   Sleep(10000);
  }
//+------------------------------------------------------------------+
//| Creates a zigzag handle and ensures readiness of its data        |
//+------------------------------------------------------------------+
bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
  {
   ResetLastError();
//--- The Zigzag indicator must be located in terminal_data_folder\MQL5\Examples
   h=iCustom(sym,tf,"Examples\\Zigzag");
   if(h==INVALID_HANDLE)
     {
      PrintFormat("%s: Failed to create the handle of the Zigzag indicator. Error code
                  __FUNCTION__,GetLastError());
      return false;
     }
//--- When creating an indicator handle, it requires time to calculate values
   int k=0; // The number of attempts to wait for the indicator calculation
//--- Wait for the calculation in a loop, pausing to 50 milliseconds if the calculati
   while(BarsCalculated(h)<=0)
     {
      k++;
      //--- Show the number of attempts
      PrintFormat("%s: k=%d",__FUNCTION__,k);
      //--- Wait 50 milliseconds to wait until the indicator is calculated
      Sleep(50);
      //--- If more than 100 attempt, then something is wrong
      if(k>100)
        {
         //--- Report a problem
         PrintFormat("Failed to calculate the indicator for %d attempts!");
         //--- Terminate the program operation
         return false;
        }
     }
//--- Everything is ready, the indicator is created and values are calculated
   return true;
  }
//+------------------------------------------------------------------+
//| Searches for the last 2 zigzag fractures and places to arrays    |
//+------------------------------------------------------------------+
bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int handle)
  {
   double values[];         // An array for the values of the zigzag
   datetime times[];        // An array to get time
   int size=100;            // Size of the array
   ResetLastError();
```

```
//--- Copy the last 100 values of the indicator
   int copied=CopyBuffer(handle,0,0,size,values);
//--- Check the number of values copied
   if(copied<100)
     {
      PrintFormat("%s: Failed to copy %d values of the indicator with the handle=%d.
                  __FUNCTION__,size,handle,GetLastError());
      return false;
     }
//--- Define the order of access to the array as in a timeseries
   ArraySetAsSeries(values,true);
//--- Write here the numbers of bars, in which fractures were found
   int positions[];
//--- Set array sizes
   ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positions,3);
//--- Counters
   int i=0,k=0;
//--- Start to search for fractures
   while(i<100)
     {
      double v=values[i];
      //--- We are not interested in empty values
      if(v!=0.0)
        {
         //--- Remember the bar number
         positions[k]=i;
         //--- Remember the value of a zigzag on the fracture
         get_values[k]=values[i];
         PrintFormat("%s: Zigzag[%d]=%G",__FUNCTION__,i,values[i]);
         //--- Increase the counter
         k++;
         //--- If two fractures found, break the loop
         if(k>2) break;
        }
      i++;
     }
//--- Define the order of access to the arrays as in a timeseries
   ArraySetAsSeries(times,true);   ArraySetAsSeries(get_times,true);
   if(CopyTime(_Symbol,_Period,0,size,times)<=0)
     {
      PrintFormat("%s: Failed to copy %d values from CopyTime(). Error code %d",
                  __FUNCTION__,size,GetLastError());
      return false;
     }
//--- Open the bar open time, on which the last 2 fractures occurred
   get_times[0]=times[positions[1]];// The last but one value will be written as the
   get_times[1]=times[positions[2]];// The value third from the end will be the secon
   PrintFormat("%s: first=%s,  second=%s",__FUNCTION__,TimeToString(get_times[1]),Time
//--- Successful
```

```
    return true;
  }
```

See also

ChartApplyTemplate(), Resources

ChartWindowFind

The function returns the number of a subwindow where an indicator is drawn. There are 2 variants of the function.

1. The function searches in the indicated chart for the subwindow with the specified "short name" of the indicator (the short name is displayed in the left top part of the subwindow), and it returns the subwindow number in case of success.

```
int  ChartWindowFind(
   long      chart_id,                 // chart identifier
   string    indicator_shortname        // short indicator name, see INDICATOR_SHORTNAM
```

2. The function must be called from a custom indicator. It returns the number of the subwindow where the indicator is working.

```
int  ChartWindowFind();
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

indicator_shortname

[in] Short name of the indicator.

Return Value

Subwindow number in case of success. In case of failure the function returns -1.

Note

If the second variant of the function (without parameters) is called from a script or Expert Advisor, the function returns -1.

Don't mix up the short name of an indicator and a file name, which is specified when an indicator is created using iCustom() and IndicatorCreate() functions. If the indicator's short name is not set explicitly, then the name of the file containing the source code of the indicator, is specified in it during compilation.

It is important to correctly form the short name of an indicator, which is recorded in the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name contains values of the indicator's input parameters, because the indicator deleted from a chart in the ChartIndicatorDelete() function is identified by its short name.

Example:

```
#property script_show_inputs
//--- input parameters
input string   shortname="MACD(12,26,9)";
//+------------------------------------------------------------------+
//| Returns number of the chart window with this indicator           |
//+------------------------------------------------------------------+
int GetIndicatorSubWindowNumber(long chartID=0,string short_name="")
```

```
  {
   int window=-1;
//---
   if((ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR)
     {
      //--- the function is called from the indicator, name is not required
      window=ChartWindowFind();
     }
   else
     {
      //--- the function is called from an Expert Advisor or script
      window=ChartWindowFind(0,short_name);
      if(window==-1) Print(__FUNCTION__+"(): Error = ",GetLastError());
     }
//---
   return(window);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int window=GetIndicatorSubWindowNumber(0,shortname);
   if(window!=-1)
      Print("Indicator "+shortname+" is in the window #"+(string)window);
   else
      Print("Indicator "+shortname+" is not found. window = "+(string)window);
  }
```

See also

ObjectCreate(), ObjectFind()

ChartTimePriceToXY

Converts the coordinates of a chart from the time/price representation to the X and Y coordinates.

```
bool  ChartTimePriceToXY(
   long           chart_id,      // Chart ID
   int            sub_window,    // The number of the subwindow
   datetime       time,          // Time on the chart
   double         price,         // Price on the chart
   int&           x,             // The X coordinate for the time on the chart
   int&           y              // The Y coordinates for the price on the chart
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart subwindow. 0 means the main chart window.

time

[in] The time value on the chart, for which the value in pixels along the X axis will be received. The origin is in the upper left corner of the main chart window.

price

[in] The price value on the chart, for which the value in pixels along the Y axis will be received. The origin is in the upper left corner of the main chart window.

x

[out] The variable, into which the conversion of time to X will be received.

y

[out] The variable, into which the conversion of price to Y will be received.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Example:

```
#define   BAR_NUMBER    0  // number of the bar we get the price and time from

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- copy data of a single bar by the BAR_NUMBER index
   MqlRates rates[]={};
   if(CopyRates(_Symbol, _Period, BAR_NUMBER, 1, rates)!=1)
```

```
      {
       PrintFormat("CopyRates() failed for bar %d. Error %d", BAR_NUMBER, GetLastError
       return;
      }

//--- convert the obtained price and time into chart pixel coordinates
   int x=0, y=0;
   ResetLastError();
   if(!ChartTimePriceToXY(ChartID(), 0, rates[0].time, rates[0].close, x, y))
      {
       Print("ChartTimePriceToXY() failed. Error ", GetLastError());
       return;
      }

//--- print the obtained result in the journal
   PrintFormat("For bar[%d] with opening time %s and price %.*f, the chart coordinates

   /*
   result:
   For bar[0] with opening time 2024.08.09 15:06 and price 1.27378, the chart coordina
   */
  }
```

See also

ChartXYToTimePrice()

ChartXYToTimePrice

Converts the X and Y coordinates on a chart to the time and price values.

```
bool  ChartXYToTimePrice(
   long           chart_id,     // Chart ID
   int            x,            // The X coordinate on the chart
   int            y,            // The Y coordinate on the chart
   int&           sub_window,   // The number of the subwindow
   datetime&      time,         // Time on the chart
   double&        price         // Price on the chart
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

x

[in] The X coordinate.

y

[in] The Y coordinate.

sub_window

[out] The variable, into which the chart subwindow number will be written. 0 means the main chart window.

time

[out] The time value on the chart, for which the value in pixels along the X axis will be received. The origin is in the upper left corner of the main chart window.

price

[out] The price value on the chart, for which the value in pixels along the Y axis will be received. The origin is in the upper left corner of the main chart window.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Show the event parameters on the chart
```

```
   Comment(__FUNCTION__,": id=",id," lparam=",lparam," dparam=",dparam," sparam=",spa
//--- If this is an event of a mouse click on the chart
   if(id==CHARTEVENT_CLICK)
     {
      //--- Prepare variables
      int      x     =(int)lparam;
      int      y     =(int)dparam;
      datetime dt    =0;
      double   price =0;
      int      window=0;
      //--- Convert the X and Y coordinates in terms of date/time
      if(ChartXYToTimePrice(0,x,y,window,dt,price))
        {
         PrintFormat("Window=%d X=%d  Y=%d  =>  Time=%s  Price=%G",window,x,y,TimeToSt
         //--- Perform reverse conversion: (X,Y) => (Time,Price)
         if(ChartTimePriceToXY(0,window,dt,price,x,y))
            PrintFormat("Time=%s  Price=%G  =>  X=%d  Y=%d",TimeToString(dt),price,x,y
         else
            Print("ChartTimePriceToXY return error code: ",GetLastError());
         //--- delete lines
         ObjectDelete(0,"V Line");
         ObjectDelete(0,"H Line");
         //--- create horizontal and vertical lines of the crosshair
         ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
         ObjectCreate(0,"V Line",OBJ_VLINE,window,dt,price);
         ChartRedraw(0);
        }
      else
         Print("ChartXYToTimePrice return error code: ",GetLastError());
      Print("+--------------------------------------------------------------+");
     }
  }
```

See also

ChartTimePriceToXY()

ChartOpen

Opens a new chart with the specified symbol and period.

```
long  ChartOpen(
   string           symbol,      // Symbol name
   ENUM_TIMEFRAMES  period       // Period
   );
```

Parameters

symbol

[in] Chart symbol. NULL means the symbol of the current chart (the Expert Advisor is attached to).

period

[in] Chart period (timeframe). Can be one of the ENUM_TIMEFRAMES values. 0 means the current chart period.

Return Value

If successful, it returns the opened chart ID. Otherwise returns 0.

Note

The maximum possible number of simultaneously open charts in the terminal can't exceed the CHARTS_MAX value.

Example:

```
#define CHART_SYMBOL  NULL
#define CHART_PERIOD  PERIOD_CURRENT

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set a new chart symbol and timeframe
   string symbol=CHART_SYMBOL;
   if(symbol==NULL)
      symbol=Symbol();
   ENUM_TIMEFRAMES timeframe = (CHART_PERIOD==PERIOD_CURRENT ? Period() : CHART_PERIOD

//--- open a new chart with the specified symbol and period
   long chart_id=ChartOpen(symbol, timeframe);
   if(chart_id==0)
     {
      Print("ChartOpen() failed. Error ", GetLastError());
      return;
     }
//--- print open chart parameters in the journal
   PrintFormat("A new chart of the %s symbol has been opened with a period of %s and
               symbol, StringSubstr(EnumToString(timeframe), 7), chart_id);
   /*
   result:
   A new chart of the GBPUSD symbol has been opened with a period of M1 and ChartID 1
   */
  }
```

ChartFirst

Returns the ID of the first chart of the client terminal.

```
long  ChartFirst();
```

Return Value

Chart ID.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the ID of the very first chart opened in the client terminal
   long            chart_id= ChartFirst();

//--- additionally get the chart symbol and period using the obtained ID
   string          symbol  = ChartSymbol(chart_id);
   ENUM_TIMEFRAMES period  = ChartPeriod(chart_id);

//--- display the description of the client terminal first chart in the journal
   PrintFormat("ID of the first chart of the client terminal: %I64d, chart symbol: %s,
   /*
   result:
   ID of the first chart of the client terminal: 133246248352168440, chart symbol: EU
   */
  }
```

ChartNext

Returns the chart ID of the chart next to the specified one.

```
long  ChartNext(
   long   chart_id      // Chart ID
   );
```

Parameters

chart_id

[in] Chart ID. 0 does not mean the current chart. 0 means "return the first chart ID".

Return Value

Chart ID. If this is the end of the chart list, it returns -1.

Example:

```
//--- variables for chart ID
   long currChart,prevChart=ChartFirst();
   int i=0,limit=100;
   Print("ChartFirst =",ChartSymbol(prevChart)," ID =",prevChart);
   while(i<limit)// We have certainly not more than 100 open charts
     {
      currChart=ChartNext(prevChart); // Get the new chart ID by using the previous c
      if(currChart<0) break;          // Have reached the end of the chart list
      Print(i,ChartSymbol(currChart)," ID =",currChart);
      prevChart=currChart;// let's save the current chart ID for the ChartNext()
      i++;// Do not forget to increase the counter
     }
```

ChartClose

Closes the specified chart.

```
bool  ChartClose(
   long   chart_id=0      // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If successful, returns true, otherwise false.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open a new chart with the same symbol and period as the current one
   long chart_id=ChartOpen(_Symbol, _Period);
   if(chart_id==0)
     {
      Print("ChartOpen() failed. Error ", GetLastError());
      return;
     }

//--- print open chart parameters in the journal
   PrintFormat("A new chart of the %s symbol has been opened with a period of %s and 
               _Symbol, StringSubstr(EnumToString(_Period), 7), chart_id);

//--- wait two seconds and close the newly opened chart
   PrintFormat("Waiting 3 seconds before closing a newly opened chart with ID %I64d .
   Sleep(3000);
   ResetLastError();
   if(!ChartClose(chart_id))
     {
      Print("ChartClose() failed. Error ", GetLastError());
      return;
     }

//--- print successful chart closure message in the journal
   PrintFormat("The chart with ID %I64d was successfully closed", chart_id);
   /*
   result:
   A new chart of the GBPUSD symbol has been opened with a period of M1 and ChartID 1
```

```
 Waiting 3 seconds before closing a newly opened chart with ID 133346697706632016 ..
 The chart with ID 133346697706632016 was successfully closed
 */
}
```

ChartSymbol

Returns the symbol name for the specified chart.

```
string  ChartSymbol(
   long  chart_id=0      // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If chart does not exist, the result will be an empty string.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart symbol and display the obtained value in the journal
   string chart_symbol = ChartSymbol();
   Print("Current chart symbol: ", chart_symbol);

//--- take the existing (in this case, the current) chart ID
   long chart_id=ChartID();
   chart_symbol=ChartSymbol(chart_id);
   PrintFormat("Chart symbol with ID %I64d: %s", chart_id, chart_symbol);

//--- set a random chart ID when receiving a symbol
   chart_symbol = ChartSymbol(1234567890);
   if(chart_symbol=="")
      Print("The chart with ID 1234567890 does not exist");
   else
      Print("Chart symbol with ID 1234567890: ", chart_symbol);
   /*
   result:
   Current chart symbol: GBPUSD
   Chart symbol with ID 132966427583395104: GBPUSD
   The chart with ID 1234567890 does not exist
   */
  }
```

See also

[ChartSetSymbolPeriod]

ChartPeriod

Returns the timeframe period of specified chart.

```
ENUM_TIMEFRAMES  ChartPeriod(
   long  chart_id=0      // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

The function returns one of the ENUM_TIMEFRAMES values. If chart does not exist, it returns 0.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart period and display the obtained value in the journal
   ENUM_TIMEFRAMES period=ChartPeriod();
   Print("Current chart period: ", EnumToString(period));

//--- take the existing (in this case, the current) chart ID
   long chart_id=ChartID();
   period=ChartPeriod(chart_id);
   PrintFormat("Chart period with ID %I64d: %s", chart_id, EnumToString(period));

//--- set a random chart ID
   period=ChartPeriod(1234567890);
   if(period==0)
      Print("The chart with ID 1234567890 does not exist");
   else
      Print("Chart period with ID 1234567890: ", EnumToString(period));
   /*
   result:
   Current chart period: PERIOD_M15
   Chart period with ID 133510090240498505: PERIOD_M15
   The chart with ID 1234567890 does not exist
   */
  }
```

ChartRedraw

This function calls a forced redrawing of a specified chart.

```
void  ChartRedraw(
   long   chart_id=0      // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Note

Usually it is used after changing the object properties.

Example:

```
#define   WIDTH      50    // rectangle width in bars

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart ID and set the graphical object name
   long   chart_id = ChartID();
   string obj_name = MQLInfoString(MQL_PROGRAM_NAME)+"_RectLabel";

//--- get two rectangle time coordinates
   datetime time1  = iTime(_Symbol, _Period, WIDTH);
   datetime time2  = iTime(_Symbol, _Period, 0);
   if(time1==0 || time2==0)
     {
      Print("Error getting time ", GetLastError());
      return;
     }

//--- get the maximum and minimum prices in the range of the rectangle width
   double price1 = HighestHigh(_Symbol, _Period, 0, WIDTH);
   double price2 = LowestLow(_Symbol, _Period, 0, WIDTH);
   if(price1==EMPTY_VALUE || price2==EMPTY_VALUE)
      return;

//--- create a rectangle object
   Print("Create a wheat-colored rectangle");
   if(!ObjectCreate(chart_id, obj_name, OBJ_RECTANGLE, 0, time1, price1, time2, price
     {
      Print("ObjectCreate() failed. Error ", GetLastError());
      return;
     }

//--- fill the rectangle with the original color
   ObjectSetInteger(chart_id, obj_name, OBJPROP_FILL, true);
   ObjectSetInteger(chart_id, obj_name, OBJPROP_BACK, true);
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrWheat);
   ChartRedraw();

//--- wait a second, fill the rectangle with the DodgerBlue color and update the chart
   Sleep(1000);
   Print("Change color to DodgerBlue");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrDodgerBlue);
   ChartRedraw();

//--- wait a second, fill the rectangle with the LimeGreen color and update the chart
   Sleep(1000);
   Print("Change color to LimeGreen");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrLimeGreen);
   ChartRedraw();

//--- wait a second, fill the rectangle with the OrangeRed color and update the chart
   Sleep(1000);
   Print("Change color to OrangeRed");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrOrangeRed);
   ChartRedraw();

//--- wait a second, fill the rectangle with the Wheat color and update the chart
   Sleep(1000);
   Print("Reset color to original");
   ObjectSetInteger(chart_id, obj_name, OBJPROP_COLOR, clrWheat);
   ChartRedraw();

//--- after three seconds, remove the object from the chart
   Sleep(3000);
   Print("Delete the rectangle");
   ObjectDelete(chart_id, obj_name);
  }
//+------------------------------------------------------------------+
//| Return the maximum High in the specified range of bars           |
//+------------------------------------------------------------------+
double HighestHigh(const string symbol, const ENUM_TIMEFRAMES timeframe, const uint st
  {
   ResetLastError();
   int index=iHighest(symbol, timeframe, MODE_HIGH, count, start);
   if(index==-1)
     {
      PrintFormat("%s: iHighest() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   GetLastError();
   double price=iHigh(symbol, timeframe, index);
   if(price==0)
     {
      PrintFormat("%s: iHigh() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   return(price);
  }
//+------------------------------------------------------------------+
//| Return the minimum Low in the specified range of bars            |
//+------------------------------------------------------------------+
double LowestLow(const string symbol, const ENUM_TIMEFRAMES timeframe, const uint star
  {
   ResetLastError();
   int index=iLowest(symbol, timeframe, MODE_LOW, count, start);
   if(index==-1)
     {
      PrintFormat("%s: iLowest() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   GetLastError();
   double price=iLow(symbol, timeframe, index);
   if(price==0)
     {
      PrintFormat("%s: iLow() failed. Error %d",__FUNCTION__, GetLastError());
      return(EMPTY_VALUE);
     }
   return(price);
  }
```

See also

[Objects functions]

ChartSetDouble

Sets a value for a corresponding property of the specified chart. Chart property should be of a double type. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool  ChartSetDouble(
   long                           chart_id,      // Chart ID
   ENUM_CHART_PROPERTY_DOUBLE     prop_id,       // Property ID
   double                         value          // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Can be one of the ENUM_CHART_PROPERTY_DOUBLE values (except the read-only properties).

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the error, call the GetLastError() function.

Note

The function is asynchronous, which means that the function does not wait for the execution of the command, which has been successfully added to the queue of specified the chart. Instead, it immediately returns control. The property will only change after the handling of the appropriate command from the chart queue. To immediately execute commands from the chart queue, call the ChartRedraw function.

If you want to immediately change several chart properties at once, then the corresponding functions (ChartSetString, ChartSetDouble, ChartSetString) should be executed in one code block, after which you should call ChartRedraw once.

To check the command execution result, you can use a function, which requests the specified chart property (ChartGetInteger, ChartGetDouble, ChartSetString). However, note that these functions are synchronous and wait for execution results.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart ID
   long chart_id = ChartID();
//--- save the initial offset of the zero bar from the right edge in percentage and th
   double shift = ChartGetDouble(chart_id, CHART_SHIFT_SIZE);
   bool   scroll= ChartGetInteger(chart_id,CHART_AUTOSCROLL);

//--- reset the auto scroll chart flag
   ChartSetInteger(chart_id, CHART_AUTOSCROLL, true);
   PrintFormat("Initial chart shift size: %.2f", shift);

//--- in the loop from 2.0% to 50.0% with the step of 0.5%
   for(int i=20; i<=500; i+=5)
     {
      //--- set the shift size as i / 10 and print the specified value in the journal
      ChartSetDouble(chart_id, CHART_SHIFT_SIZE, i/10.0);
      PrintFormat("Set chart shift size to %.1f%%", i/10.0);

      //--- keep the chart zero bar at the specified shift distance
      ChartNavigate(chart_id, CHART_END, 0);

      //--- wait a bit
      Sleep(16);
     }

//--- set the initial chart shift and auto scroll
   Print("Set the chart shift size to the initial value of ", shift);
   ChartSetDouble(ChartID(), CHART_SHIFT_SIZE, shift);
   ChartSetInteger(chart_id, CHART_AUTOSCROLL, scroll);
  }
```

ChartSetInteger

Sets a value for a corresponding property of the specified chart. Chart property must be datetime, int, color, bool or char. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool  ChartSetInteger(
   long                            chart_id,      // Chart ID
   ENUM_CHART_PROPERTY_INTEGER     prop_id,       // Property ID
   long                            value          // Value
   );
```

Sets a value for a corresponding property of the specified subwindow.

```
bool  ChartSetInteger(
   long                            chart_id,      // Chart ID
   ENUM_CHART_PROPERTY_INTEGER     prop_id,       // Property ID
   int                             sub_window,    // Subwindow number
   long                            value          // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. It can be one of the ENUM_CHART_PROPERTY_INTEGER value (except the read-only properties).

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the error, call the GetLastError() function.

Note

The function is asynchronous, which means that the function does not wait for the execution of the command, which has been successfully added to the queue of specified the chart. Instead, it immediately returns control. The property will only change after the handling of the appropriate command from the chart queue. To immediately execute commands from the chart queue, call the ChartRedraw function.

If you want to immediately change several chart properties at once, then the corresponding functions (ChartSetString, ChartSetDouble, ChartSetString) should be executed in one code block, after which you should call ChartRedraw once.

To check the command execution result, you can use a function, which requests the specified chart property (ChartGetInteger, ChartGetDouble, ChartSetString). However, note that these functions are synchronous and wait for execution results.

Example:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Enabling events of mouse movements on the chart window
   ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
//--- Forced updating of chart properties ensures readiness for event processing
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| MouseState                                                       |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // mouse left
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // mouse right
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // mouse middle
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // mouse first X key
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // mouse second X key
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // shift key
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // control key
   return(res);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
  {
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)sparam));
  }
```

ChartSetString

Sets a value for a corresponding property of the specified chart. Chart property must be of the string type. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool  ChartSetString(
   long                          chart_id,     // Chart ID
   ENUM_CHART_PROPERTY_STRING    prop_id,      // Property ID
   string                        str_value     // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Its value can be one of the ENUM_CHART_PROPERTY_STRING values (except the read-only properties).

str_value

[in] Property value string. String length cannot exceed 2045 characters (extra characters will be truncated).

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the error, call the GetLastError() function.

Note

ChartSetString can be used for a comment output on the chart instead of the Comment function.

The function is asynchronous, which means that the function does not wait for the execution of the command, which has been successfully added to the queue of specified the chart. Instead, it immediately returns control. The property will only change after the handling of the appropriate command from the chart queue. To immediately execute commands from the chart queue, call the ChartRedraw function.

If you want to immediately change several chart properties at once, then the corresponding functions (ChartSetString, ChartSetDouble, ChartSetString) should be executed in one code block, after which you should call ChartRedraw once.

To check the command execution result, you can use a function, which requests the specified chart property (ChartGetInteger, ChartGetDouble, ChartSetString). However, note that these functions are synchronous and wait for execution results.

Example:

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
   string comment=StringFormat("Display prices:\nAsk = %G\nBid = %G\nSpread = %d",
                               Ask,Bid,Spread);
   ChartSetString(0,CHART_COMMENT,comment);
  }
```

See also

Comment, ChartGetString

ChartGetDouble

Returns the value of a corresponding property of the specified chart. Chart property must be of double type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
double  ChartGetDouble(
   long                          chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,           // Property ID
   int                           sub_window=0       // subwindow number, if necessary
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable double_var passed by reference.

```
bool  ChartGetDouble(
   long                          chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_DOUBLE    prop_id,           // Property ID
   int                           sub_window,      // Subwindow number
   double&                       double_var       // Target variable for the chart prop
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_DOUBLE values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

double_var

[out] Target variable of double type for the requested property.

Return Value

The value of double type.

For the second call case it returns true if the specified property is available and its value has been placed into double_var variable, otherwise returns false. To get an additional information about the error, it is necessary to call the function GetLastError().

Note

The function is synchronous, which means that it waits for the execution of all the commands that have been added to the chart queue prior to its call.

Example:

```
void OnStart()
```

```
  {
   double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,0);
   double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,0);
   Print("CHART_PRICE_MIN =",priceMin);
   Print("CHART_PRICE_MAX =",priceMax);
  }
```

ChartGetInteger

Returns the value of a corresponding property of the specified chart. Chart property must be of datetime, int or bool type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
long  ChartGetInteger(
   long                          chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_INTEGER   prop_id,           // Property ID
   int                           sub_window=0       // subwindow number, if necessary
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable long_var passed by reference.

```
bool  ChartGetInteger(
   long                          chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_INTEGER   prop_id,           // Property ID
   int                           sub_window=0       // subwindow number
   long&                         long_var           // Target variable for the propert
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_INTEGER values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

long_var

[out] Target variable of long type for the requested property.

Return Value

The value of long type.

For the second call case it returns true if specified property is available and its value has been stored into long_var variable, otherwise returns false. To get additional information about the error, it is necessary to call the function GetLastError().

Note

The function is synchronous, which means that it waits for the execution of all the commands that have been added to the chart queue prior to its call.

Example:

```
void OnStart()
```

```
  {
   int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   Print("CHART_HEIGHT_IN_PIXELS =",height,"pixels");
   Print("CHART_WIDTH_IN_PIXELS =",width,"pixels");
  }
```

ChartGetString

Returns the value of a corresponding property of the specified chart. Chart property must be of string type. There are 2 variants of the function call.

1. Returns the property value directly.

```
string  ChartGetString(
   long                            chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_STRING      prop_id            // Property ID
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable string_var passed by reference.

```
bool  ChartGetString(
   long                            chart_id,          // Chart ID
   ENUM_CHART_PROPERTY_STRING      prop_id,           // Property ID
   string&                         string_var         // Target variable for the property
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_STRING values.

string_var

[out] Target variable of string type for the requested property.

Return Value

The value of string type.

For the second call case it returns true if the specified property is available and its value has been stored into string_var variable, otherwise returns false. To get additional information about the error, it is necessary to call the function GetLastError().

Note

ChartGetString can be used for reading comments plotted on the chart using the Comment or ChartSetString functions.

The function is synchronous, which means that it waits for the execution of all the commands that have been added to the chart queue prior to its call.

Example:

```
void OnStart()
  {
   ChartSetString(0,CHART_COMMENT,"Test comment.\nSecond line.\nThird!");
   ChartRedraw();
   Sleep(1000);
   string comm=ChartGetString(0,CHART_COMMENT);
   Print(comm);
  }
```

See also

Comment, ChartSetString

ChartNavigate

Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart.

```
bool  ChartNavigate(
   long                     chart_id,      // Chart ID
   ENUM_CHART_POSITION      position,      // Position
   int                      shift=0        // Shift value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

position

[in] Chart position to perform a shift. Can be one of the ENUM_CHART_POSITION values.

shift=0

[in] Number of bars to shift the chart. Positive value means the right shift (to the end of chart), negative value means the left shift (to the beginning of chart). The zero shift can be used to navigate to the beginning or end of chart.

Return Value

Returns true if successful, otherwise returns false.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get handle of the current chart
   long handle=ChartID();
   string comm="";
   if(handle>0) // if successful, additionally set up the chart
     {
      //--- disable auto scroll
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- set a shift from the right chart border
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- draw candlesticks
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- set the display mode for tick volumes
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

      //--- prepare a text to output in Comment()
      comm="Scroll 10 bars to the right of the history start";
      //--- show comment
```

```
 Comment(comm);
 //--- scroll 10 bars to the right of the history start
 ChartNavigate(handle,CHART_BEGIN,10);
 //--- get the number of the first bar visible on the chart (numeration like in
 long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
 //--- add line feed character
 comm=comm+"\r\n";
 //--- add to comment
 comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\
 //--- show comment
 Comment(comm);
 //--- wait 5 seconds to see how the chart moves
 Sleep(5000);

 //--- add to the comment text
 comm=comm+"\r\n"+"Scroll 10 bars to the left of the right chart border";
 Comment(comm);
 //--- scroll 10 bars to the left of the right chart border
 ChartNavigate(handle,CHART_END,-10);
 //--- get the number of the first bar visible on the chart (numeration like in
 first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
 comm=comm+"\r\n";
 comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\
 Comment(comm);
 //--- wait 5 seconds to see how the chart moves
 Sleep(5000);

 //--- new block of chart scrolling
 comm=comm+"\r\n"+"Scroll 300 bars to the right of the history start";
 Comment(comm);
 //--- scroll 300 bars to the right of the history start
 ChartNavigate(handle,CHART_BEGIN,300);
 first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
 comm=comm+"\r\n";
 comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\
 Comment(comm);
 //--- wait 5 seconds to see how the chart moves
 Sleep(5000);

 //--- new block of chart scrolling
 comm=comm+"\r\n"+"Scroll 300 bars to the left of the right chart border";
 Comment(comm);
 //--- scroll 300 bars to the left of the right chart border
 ChartNavigate(handle,CHART_END,-300);
 first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
 comm=comm+"\r\n";
 comm=comm+"The first bar on the chart is number "+IntegerToString(first_bar)+"\
 Comment(comm);
}
```

```
  }
```

ChartID

Returns the ID of the current chart.

```
long  ChartID();
```

Return Value

Value of long type.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- variables for chart identifiers
   long curr_chart=ChartFirst();
   int i=0;
   //--- print the first chart data in the journal
   PrintFormat("Chart[%d] ID: %I64d,  symbol: %s", i, curr_chart, ChartSymbol(curr_cha

//--- until the open chart limit is reached (CHARTS_MAX)
   while(!IsStopped() && i < CHARTS_MAX)
     {
      //--- increase the chart counter
      i++;
      //--- get the next chart ID based on the previous one
      curr_chart=ChartNext(curr_chart);

      //--- terminate the loop if the end of the chart list is reached
      if(curr_chart<0)
         break;

      //--- print the next chart data in the journal
      PrintFormat("Chart[%d] ID: %I64d,  symbol: %s", i, curr_chart, ChartSymbol(curr_
     }
   /*
   result:
   Chart[0] ID: 133246248352168440,  symbol: EURUSD
   Chart[1] ID: 133346697706632015,  symbol: USDJPY
   Chart[2] ID: 133246248352168439,  symbol: GBPUSD
   Chart[3] ID: 133346697706632009,  symbol: RU000A103661
   Chart[4] ID: 133346697706632010,  symbol: AEM4
   Chart[5] ID: 133346697706632011,  symbol: AA.SPB
   Chart[6] ID: 133346697706632012,  symbol: ALLFUTMIX
   Chart[7] ID: 133346697706632013,  symbol: EURUSD
   Chart[8] ID: 133346697706632014,  symbol: SBER
   */
  }
```

ChartIndicatorAdd

Adds an indicator with the specified handle into a specified chart window. Indicator and chart should be generated on the same symbol and time frame.

```
bool  ChartIndicatorAdd(
   long   chart_id,                 // chart ID
   int    sub_window                // number of the sub-window
   int    indicator_handle          // handle of the indicator
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart sub-window. 0 means the main chart window. To add an indicator in a new window, the parameter must be one greater than the index of the last existing window, i.e. equal to CHART_WINDOWS_TOTAL. If the value of the parameter is greater than CHART_WINDOWS_TOTAL, a new window will not be created, and the indicator will not be added.

indicator_handle

[in] The handle of the indicator.

Return Value

The function returns true in case of success, otherwise it returns false. In order to obtain information about the error, call the GetLastError() function. Error 4114 means that a chart and an added indicator differ by their symbol or time frame.

Note

If an indicator that should be drawn in a separate subwindow (for example, built-in iMACD or a custom indicator with specified #property indicator_separate_window property) is applied to the main chart window, it may not be visible though it will still be present in the list of indicators. This means that the scale of the indicator is different from the scale of the price chart, and applied indicator's values do not fit in the displayed range of the price chart. In this case, GetLastError() returns zero code indicating the absence of an error. The values of such "invisible" indicator can be seen in Data Window and received from other MQL5 applications.

Example:

```
#property description "Expert Advisor demonstrating the work with ChartIndicatorAdd()
#property description "After launching on the chart (and receiving the error in Journa
#property description "the Expert Advisor's properties and specify correct <symbol> ar
#property description "MACD indicator will be added on the chart."

//--- input parameters
input string          symbol="AUDUSD";    // symbol name
input ENUM_TIMEFRAMES period=PERIOD_M12;  // time frame
input int     fast_ema_period=12;         // fast MACD period
input int     slow_ema_period=26;         // slow MACD period
input int      signal_period=9;           // signal period
input ENUM_APPLIED_PRICE apr=PRICE_CLOSE; // price type for MACD calculation

int indicator_handle=INVALID_HANDLE;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_period,
//--- try to add the indicator on the chart
   if(!AddIndicator())
     {
      //--- AddIndicator() function refused to add the indicator on the chart
      int answer=MessageBox("Do you want to add MACD on the chart anyway?",
                            "Incorrect symbol and/or time frame for adding the indicat
                            MB_YESNO // "Yes" and "No" selection buttons will be show
                            );
      //--- if a user still insists on incorrect usage of ChartIndicatorAdd()
      if(answer==IDYES)
        {
         //--- first of all, a Journal entry will be made about that
         PrintFormat("Attention! %s: Trying to add MACD(%s/%s) indicator on %s/%s cha
                     __FUNCTION__,symbol,EnumToString(period),_Symbol,EnumToString(_Pe
         //--- receive the number of a new subwindow, to which we will try to add the
         int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
         //--- now make an attempt resulting in error
         if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
            PrintFormat("Failed to add MACD indicator on %d chart window. Error code
                        subwindow,GetLastError());
        }
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
// Expert Advisor performs nothing
  }
//+------------------------------------------------------------------+
//| Function for checking and adding the indicator on the chart      |
//+------------------------------------------------------------------+
bool AddIndicator()
  {
//--- displayed message
```

```
   string message;
//--- check if the indicator symbol and chart symbol match each other
   if(symbol!=_Symbol)
     {
      message="Displaying the use of Demo_ChartIndicatorAdd() function:";
      message=message+"\r\n";
      message=message+"Unable to add the indicator calculated on another symbol on the
      message=message+"\r\n";
      message=message+"Specify the chart symbol in Expert Advisor's property - "+_Sym
      Alert(message);
      //--- premature exit, the indicator will not be added on the chart
      return false;
     }
//--- check if the indicator's and chart's time frames match each other
   if(period!=_Period)
     {
      message="Unable to add the indicator calculated on another time frame on the cha
      message=message+"\r\n";
      message=message+"Specify the chart time frame in Expert Advisor properties - "+
      Alert(message);
      //--- premature exit, the indicator will not be added on the chart
      return false;
     }
//--- all checks completed, symbol and indicator time frame match the chart
   if(indicator_handle==INVALID_HANDLE)
     {
      Print(__FUNCTION__,"  Creating MACD indicator");
      indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_per
      if(indicator_handle==INVALID_HANDLE)
        {
         Print("Failed to create MACD indicator. Error code ",GetLastError());
        }
     }
//--- reset the error code
   ResetLastError();
//--- apply the indicator to the chart
   Print(__FUNCTION__,"  Adding MACD indicator on the chart");
   Print("MACD is generated on ",symbol,"/",EnumToString(period));
//--- receive the number of a new subwindow, to which MACD indicator is added
   int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   PrintFormat("Adding MACD indicator on %d chart window",subwindow);
   if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
     {
      PrintFormat("Failed to add MACD indicator on %d chart window. Error code  %d",
                  subwindow,GetLastError());
     }
//--- Indicator added successfully
   return(true);
  }
```

See Also

ChartIndicatorDelete(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate()

ChartIndicatorDelete

Removes an indicator with a specified name from the specified chart window.

```
bool  ChartIndicatorDelete(
   long           chart_id,              // chart id
   int            sub_window             // number of the subwindow
   const string   indicator_shortname    // short name of the indicator
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const indicator_shortname

[in] The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get the short name of an indicator use the ChartIndicatorName() function.

Return Value

Returns true in case of successful deletion of the indicator. Otherwise it returns false. To get error details use the GetLastError() function.

Note

If two indicators with identical short names exist in the chart subwindow, the first one in a row will be deleted.

If other indicators on this chart are based on the values of the indicator that is being deleted, such indicators will also be deleted.

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

Deletion of an indicator from a chart doesn't mean that its calculation part will be deleted from the terminal memory. To release the indicator handle use the IndicatorRelease() function.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the ChartIndicatorDelete() function is identified by the short name.

Example of deleting an indicator after initialization has failed:

```
//+------------------------------------------------------------------+
//|                                     Demo_ChartIndicatorDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                                 https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      first_param=1;
input int      second_param=2;
input int      third_param=3;
input bool     wrong_init=true;
//--- indicator buffers
double         HistogramBuffer[];
string         shortname;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int res=INIT_SUCCEEDED;
//--- Link the HistogramBuffer array to the indicator buffer
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- Construct a short indicator name based on input parameters
   shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                          first_param,second_param,third_param);
   IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- If forced completion of an indicator is set, return a non-zero value
   if(wrong_init) res=INIT_FAILED;
   return(res);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                  const long &volume[],
                  const int &spread[])
  {
//--- Starting position for working in a loop
   int start=prev_calculated-1;
   if(start<0) start=0;
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      HistogramBuffer[i]=close[i];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| A handler of the Deinit event                                    |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   PrintFormat("%s: Deinitialization reason code=%d",__FUNCTION__,reason);
   if(reason==REASON_INITFAILED)
     {
      PrintFormat("An indicator with a short name %s (file %s) deletes itself from the
      int window=ChartWindowFind();
      bool res=ChartIndicatorDelete(0,window,shortname);
      //--- Analyse the result of call of ChartIndicatorDelete()
      if(!res)
        {
         PrintFormat("Failed to delete indicator %s from window #%d. Error code %d",
                     shortname,window,GetLastError());
        }
     }
  }
```

See also

ChartIndicatorAdd(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartIndicatorGet

Returns the handle of the indicator with the specified short name in the specified chart window.

```
int  ChartIndicatorGet(
   long           chart_id,              // Chart ID
   int            sub_window             // The number of the subwindow
   const string   indicator_shortname    // Short name of the indicator
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart subwindow. 0 means the main chart window.

const indicator_shortname

[in] The short name if the indicator, which is set in the INDICATOR_SHORTNAME property using the IndicatorSetString() function. To get the short name of an indicator, use the ChartIndicatorName() function.

Return Value

Returns an indicator handle if successful, otherwise returns INVALID_HANDLE. To get information about the error, call the GetLastError() function.

Note

The indicator handle obtained using the ChartIndicatorGet() function increases the internal indicator usage counter. The terminal runtime system keeps all indicators, whose counter is greater than zero, loaded. Therefore, the indicator handle that is no longer needed should be immediately and explicitly released using IndicatorRelease() in the same program that received it, as shown in the example below. Otherwise, it will be impossible to find the “abandoned” handle and release it from another program correctly.

When creating an indicator, be careful forming its short name, which is written in the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that a short name should contain the values of input parameters of the indicator, since the indicator is identified in the ChartIndicatorGet() function based on its short name.

Another way to identify the indicator is to get a list of its parameters for a given handle using the IndicatorParameters() function and then to analyze the obtained values.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- The number of windows on the chart (at least one main window is always presen
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   //--- Check all windows
   for(int w=0;w<windows;w++)
     {
      //--- the number of indicators in this window/subwindow
      int total=ChartIndicatorsTotal(0,w);
      //--- Go through all indicators in the window
      for(int i=0;i<total;i++)
        {
         //--- get the short name of an indicator
         string name=ChartIndicatorName(0,w,i);
         //--- get the handle of an indicator
         int handle=ChartIndicatorGet(0,w,name);
         //--- Add to log
         PrintFormat("Window=%d,  index=%d,  name=%s,  handle=%d",w,i,name,handle);
         //--- You should obligatorily release the indicator handle when it is no long
         IndicatorRelease(handle);
        }
     }
  }
```

See also

[ChartIndicatorAdd()](), [ChartIndicatorName()](), [ChartIndicatorsTotal()](), [IndicatorParameters()]()

ChartIndicatorName

Returns the short name of the indicator by the number in the indicators list on the specified chart window.

```
string  ChartIndicatorName(
   long  chart_id,       // chart id
   int   sub_window      // number of the subwindow
   int   index           // index of the indicator in the list of indicators added to t
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

index

[in] the index of the indicator in the list of indicators. The numeration of indicators start with zero, i.e. the first indicator in the list has the 0 index. To obtain the number of indicators in the list use the ChartIndicatorsTotal() function.

Return Value

The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get error details use the GetLastError() function.

Note

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the ChartIndicatorDelete() function is identified by the short name.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart ID and the number of its subwindows, including the main wi
   long chart_id = ChartID();
   int  wnd_total= (int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);

//--- iterate over all current chart windows in a loop
   for(int w=0; w<wnd_total; w++)
     {
      //--- get the number of indicators attached to the chart window specified by the
      int ind_total=ChartIndicatorsTotal(chart_id, w);

      //--- print the header for the selected chart window
      PrintFormat("Chart window %d indicators: ", w);

      //--- get in the loop and write to the variable all the names of indicators atta
      string ind_names="";
      for(int i=0; i<ind_total; i++)
        {
         ind_names+="  "+ChartIndicatorName(chart_id, w, i)+(i<ind_total-1 ? "\n": "")
        }
      //--- print the obtained list of names of all indicators, attached to the speci
      Print(ind_names);
     }
   /*
   result:
   Chart window 0 indicators:
     AMA(9,2,30)
     SAR(0.02,0.20)
     Fractals
   Chart window 1 indicators:
     RSI(14)
     AMA(9,2,30)
   Chart window 2 indicators:
     MFI(14)
   */
  }
```

See also

ChartIndicatorAdd(), ChartIndicatorDelete(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartIndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int  ChartIndicatorsTotal(
   long   chart_id,      // chart id
   int    sub_window     // number of the subwindow
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

Return Value

The number of indicators in the specified chart window. To get error details use the GetLastError() function.

Note

The function allows going searching through all the indicators attached to the chart. The number of all the windows of the chart can be obtained from the CHART_WINDOWS_TOTAL property using the ChartGetInteger() function.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart ID and the number of its subwindows, including the main w
   long chart_id = ChartID();
   int  wnd_total= (int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);

//--- iterate over all current chart windows in a loop
   for(int w=0; w<wnd_total; w++)
     {
      //--- get the number of indicators attached to the chart window specified by the
      int ind_total=ChartIndicatorsTotal(chart_id, w);

      //--- get the number of indicators attached to the specified chart window
      PrintFormat("Chart ID %I64d, subwindow %d. Attached indicators: %d", chart_id, w
     }
   /*
   result:
   Chart ID 133246248352168439, subwindow 0. Attached indicators: 3
   Chart ID 133246248352168439, subwindow 1. Attached indicators: 2
```

```
    Chart ID 133246248352168439, subwindow 2. Attached indicators: 1
    */
  }
```

See also

ChartIndicatorAdd(), ChartIndicatorDelete(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartWindowOnDropped

Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to. 0 means the main chart window.

```
int  ChartWindowOnDropped();
```

Return Value

Value of int type.

Example:

```
   int myWindow=ChartWindowOnDropped();
   int windowsTotal=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   Print("Script is running on the window #"+myWindow+
         ". Total windows on the chart "+ChartSymbol()+":",windowsTotal);
```

See also

ChartPriceOnDropped, ChartTimeOnDropped, ChartXOnDropped, ChartYOnDropped

ChartPriceOnDropped

Returns the price coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
double  ChartPriceOnDropped();
```

Return Value

Value of double type.

Example:

```
   double p=ChartPriceOnDropped();
   Print("ChartPriceOnDropped() = ",p);
```

See also

ChartXOnDropped, ChartYOnDropped

ChartTimeOnDropped

Returns the time coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
datetime  ChartTimeOnDropped();
```

Return Value

Value of datetime type.

Example:

```
   datetime t=ChartTimeOnDropped();
   Print("Script was dropped on the "+t);
```

See also

ChartXOnDropped, ChartYOnDropped

ChartXOnDropped

Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to.

```
int  ChartXOnDropped();
```

Return Value

The X coordinate value.

Note

X axis direction from left to right.

Example:

```
   int X=ChartXOnDropped();
   int Y=ChartYOnDropped();
   Print("(X,Y) = ("+X+","+Y+")");
```

See also

ChartWindowOnDropped, ChartPriceOnDropped, ChartTimeOnDropped

ChartYOnDropped

Returns the Y coordinateof the chart point the Expert Advisor or script has been dropped to.

```
int  ChartYOnDropped();
```

Return Value

The Y coordinate value.

Note

Y axis direction from top to bottom.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get and print the X and Y coordinates of the chart point the script was dropped
   int x=ChartXOnDropped();
   int y=ChartYOnDropped();
   PrintFormat("Script dropped to coordinates X = %d, Y = %d", x, y);
   /*
   result:
   Script dropped to coordinates X = 429, Y = 114
   */
  }
```

See also

[ChartWindowOnDropped](), [ChartPriceOnDropped](), [ChartTimeOnDropped]()

ChartSetSymbolPeriod

Changes the symbol and period of the specified chart. The function is asynchronous, i.e. it sends the command and does not wait for its execution completion. The command is added to chart messages queue and will be executed after processing of all previous commands.

```
bool  ChartSetSymbolPeriod(
   long              chart_id,      // Chart ID
   string            symbol,        // Symbol name
   ENUM_TIMEFRAMES   period         // Period
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

symbol

[in] Chart symbol. NULL value means the current chart symbol (Expert Advisor is attached to)

period

[in] Chart period (timeframe). Can be one of the ENUM_TIMEFRAMES values. 0 means the current chart period.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get an information about the error, call the GetLastError() function.

Note

The symbol/period change leads to the re-initialization of the Expert Advisor attached to a chart.

The call of ChartSetSymbolPeriod with the same symbol and timeframe can be used to update the chart (similar to the terminal's Refresh command). In its turn, the chart update triggers re-calculation of the indicators attached to it. Thus, you are able to calculate an indicator on the chart even if there are no ticks (e.g., on weekends).

Example:

```
#define   SYMBOL    "GBPUSD"
#define   PERIOD    PERIOD_H1

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the current chart ID, symbol and period
   long              chart_id= ChartID();
   string            symbol   = Symbol();
   ENUM_TIMEFRAMES period   = Period();
```

```
//--- report in the journal on replacing the current chart symbol and period with the
   PrintFormat("Change the %s symbol and the %s period of the chart %I64u to %s %s",
               symbol, TimeframeDescrioption(period), chart_id, SYMBOL, TimeframeDesc

//--- change the chart symbol and period
   ChartSetSymbolPeriod(chart_id, SYMBOL, PERIOD);
   /*
   result:
   Change the EURUSD symbol and the M1 period of the chart 133246248352168440 to GBPUS
   */
  }
//+------------------------------------------------------------------+
//| Return the chart period description                              |
//+------------------------------------------------------------------+
string TimeframeDescrioption(const ENUM_TIMEFRAMES timeframe)
  {
   return(StringSubstr(EnumToString(timeframe), 7));
  }
```

See also

ChartSymbol, ChartPeriod

ChartScreenShot

The function provides a screenshot of the chart in its current state in the GIF, PNG or BMP format depending on specified extension.

```
bool  ChartScreenShot(
   long               chart_id,                    // Chart ID
   string             filename,                    // Symbol name
   int                width,                       // Width
   int                height,                      // Height
   ENUM_ALIGN_MODE    align_mode=ALIGN_RIGHT       // Alignment type
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] Screenshot file name. Cannot exceed 63 characters. Screenshot files are placed in the \Files directory.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Output mode of a narrow screenshot. A value of the ENUM_ALIGN_MODE enumeration. ALIGN_RIGHT means align to the right margin (the output from the end). ALIGN_LEFT means Left justify.

Return Value

Returns true if successful, otherwise false.

Note

If you need to take a screenshot from a chart from a certain position, first it's necessary to position the graph using the ChartNavigate() function. If the horizontal size of the screenshot is smaller than the chart window, either the right part of the chart window, or its left part is output, depending on the align_mode settings.

Example:

```
#property description "The Expert Advisor demonstrates how to create a series of scree
#property description "chart using the ChartScreenShot() function. For convenience, th
#property description "shown on the chart. The height and width of images is defined t

#define        WIDTH  800     // Image width to call ChartScreenShot()
#define        HEIGHT 600     // Image height to call ChartScreenShot()
```

```
//--- input parameters
input int      pictures=5;    // The number of images in the series
int            mode=-1;       // -1 denotes a shift to the right edge of the chart, 1
int            bars_shift=300;// The number of bars when scrolling the chart using Cha
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Disable chart autoscroll
   ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- Set the shift of the right edge of the chart
   ChartSetInteger(0,CHART_SHIFT,true);
//--- Show a candlestick chart
   ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
   Print("Preparation of the Expert Advisor is completed");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Show the name of the function, call time and event identifier
   Print(__FUNCTION__,TimeCurrent(),"   id=",id,"   mode=",mode);
//--- Handle the CHARTEVENT_CLICK event ("A mouse click on the chart")
   if(id==CHARTEVENT_CLICK)
     {
      //--- Initial shift from the chart edge
      int pos=0;
      //--- Operation with the left chart edge
      if(mode>0)
        {
         //--- Scroll the chart to the left edge
         ChartNavigate(0,CHART_BEGIN,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- Prepare a text to show on the chart and a file name
            string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif";
            //--- Show the name on the chart as a comment
            Comment(name);
            //--- Save the chart screenshot in a file in the terminal_directory\MQL5\
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- Give the user time to look at the new part of the chart
            Sleep(3000);
            //--- Scroll the chart from the current position bars_shift bars to the r
            ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
           }
         //--- Change the mode to the opposite
         mode*=-1;
        }
      else // Operation with the right chart edge
        {
         //--- Scroll the chart to the right edge
         ChartNavigate(0,CHART_END,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- Prepare a text to show on the chart and a file name
            string name="ChartScreenShot"+"CHART_END"+string(pos)+".gif";
            //--- Show the name on the chart as a comment
            Comment(name);
            //--- Save the chart screenshot in a file in the terminal_directory\MQL5\
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_RIGHT))
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- Give the user time to look at the new part of the chart
            Sleep(3000);
            //--- Scroll the chart from the current position bars_shift bars to the r
            ChartNavigate(0,CHART_CURRENT_POS,-bars_shift);
           }
         //--- Change the mode to the opposite
         mode*=-1;
        }
     }  // End of CHARTEVENT_CLICK event handling
//--- End of the OnChartEvent() handler
  }
```

See also

[ChartNavigate()](), [Resources]()

Trade Functions

This is the group of functions intended for managing trading activities.

Before you proceed to study the trade functions of the platform, you must have a clear understanding of the basic terms: order, deal and position:

- An order is an instruction given to a broker to buy or sell a financial instrument. There are two main types of orders: Market and Pending. In addition, there are special Take Profit and Stop Loss levels.
- A deal is the commercial exchange (buying or selling) of a financial security. Buying is executed at the demand price (Ask), and Sell is performed at the supply price (Bid). A deal can be opened as a result of market order execution or pending order triggering. Note that in some cases, execution of an order can result in several deals.
- A position is a trade obligation, i.e. the number of bought or sold contracts of a financial instrument. A long position is financial security bought expecting the security price go higher. A short position is an obligation to supply a security expecting the price will fall in future.

General information about trading operations is available in the client terminal help.

Trading functions can be used in Expert Advisors and scripts. Trading functions can be called only if in the properties of the Expert Advisor or script the "Allow live trading" checkbox is enabled.

Trading can be allowed or prohibited depending on various factors described in the Trade Permission section.

Function	Action
OrderCalcMargin	Calculates the margin required for the specified order type, in the deposit currency
OrderCalcProfit	Calculates the profit based on the parameters passed, in the deposit currency
OrderCheck	Checks if there are enough funds to execute the required trade operation.
OrderSend	Sends trade requests to a server
OrderSendAsync	Asynchronously sends trade requests without waiting for the trade response of the trade server
PositionsTotal	Returns the number of open positions
PositionGetSymbol	Returns the symbol corresponding to the open position
PositionSelect	Chooses an open position for further working with it
PositionSelectByTicket	Selects a position to work with by the ticket number specified in it
PositionGetDouble	Returns the requested property of an open position (double)
PositionGetInteger	Returns the requested property of an open position (datetime or int)
PositionGetString	Returns the requested property of an open position (string)

Function	Action
PositionGetTicket	Returns the ticket of the position with the specified index in the list of open positions
OrdersTotal	Returns the number of orders
OrderGetTicket	Return the ticket of a corresponding order
OrderSelect	Selects a order for further working with it
OrderGetDouble	Returns the requested property of the order (double)
OrderGetInteger	Returns the requested property of the order (datetime or int)
OrderGetString	Returns the requested property of the order (string)
HistorySelect	Retrieves the history of transactions and orders for the specified period of the server time
HistorySelectByPosition	Requests the history of deals with a specified position identifier.
HistoryOrderSelect	Selects an order in the history for further working with it
HistoryOrdersTotal	Returns the number of orders in the history
HistoryOrderGetTicket	Return order ticket of a corresponding order in the history
HistoryOrderGetDouble	Returns the requested property of an order in the history (double)
HistoryOrderGetInteger	Returns the requested property of an order in the history (datetime or int)
HistoryOrderGetString	Returns the requested property of an order in the history (string)
HistoryDealSelect	Selects a deal in the history for further calling it through appropriate functions
HistoryDealsTotal	Returns the number of deals in the history
HistoryDealGetTicket	Returns a ticket of a corresponding deal in the history
HistoryDealGetDouble	Returns the requested property of a deal in the history (double)
HistoryDealGetInteger	Returns the requested property of a deal in the history (datetime or int)
HistoryDealGetString	Returns the requested property of a deal in the history (string)

OrderCalcMargin

The function calculates the margin required for the specified order type, on the current account, in the current market environment not taking into account current pending orders and open positions. It allows the evaluation of margin for the trade operation planned. The value is returned in the account currency.

```
bool  OrderCalcMargin(
   ENUM_ORDER_TYPE       action,           // type of order
   string                symbol,           // symbol name
   double                volume,           // volume
   double                price,            // open price
   double&               margin            // variable for obtaining the margin value
   );
```

Parameters

action

[in] The order type, can be one of the values of the ENUM_ORDER_TYPE enumeration.

symbol

[in] Symbol name.

volume

[in] Volume of the trade operation.

price

[in] Open price.

margin

[out] The variable, to which the value of the required margin will be written in case the function is successfully executed. The calculation is performed as if there were no pending orders and open positions on the current account. The margin value depends on many factors, and can differ in different market environments.

Return Value

The function returns true in case of success; otherwise it returns false. In order to obtain information about the error, call the GetLastError() function.

Example:

```
#define   VOLUME     1.0   // order volume
#define   DEVIATION  100   // distance for setting a pending order
#define   STOP_LIMIT 50    // order StopLimit distance

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   int array_types[8]={ORDER_TYPE_BUY,
                       ORDER_TYPE_SELL,
                       ORDER_TYPE_BUY_LIMIT,
                       ORDER_TYPE_SELL_LIMIT,
                       ORDER_TYPE_BUY_STOP,
                       ORDER_TYPE_SELL_STOP,
                       ORDER_TYPE_BUY_STOP_LIMIT,
                       ORDER_TYPE_SELL_STOP_LIMIT };

//--- in a loop by the order type array
   for(int i=0; i<(int)array_types.Size(); i++)
     {
      //--- depending on the order type, get ORDER_TYPE_BUY or ORDER_TYPE_SELL type
      ENUM_ORDER_TYPE type=MarketOrderByOrderType((ENUM_ORDER_TYPE)array_types[i]);

      //--- depending on the order type, get the price
      double price=PriceByOrderType(_Symbol, type);

      //--- get the amount of margin required for the order type specified in 'action
      double margin=EMPTY_VALUE;
      ResetLastError();
      if(!OrderCalcMargin(type, _Symbol, VOLUME, price, margin))
        {
         Print("OrderCalcMargin() failed. Error ",GetLastError());
         continue;
        }
      //--- print the result in the journal
      PrintFormat("Margin required for %.2f %s order at price %.*f on %s symbol: %.2f
     }
   /*
   result:
   Margin required for 1.00 Buy order at price 1.31668 on GBPUSD symbol: 1316.68 USD
   Margin required for 1.00 Sell order at price 1.31661 on GBPUSD symbol: 1316.61 USD
   Margin required for 1.00 Buy Limit order at price 1.31568 on GBPUSD symbol: 1315.6
   Margin required for 1.00 Sell Limit order at price 1.31761 on GBPUSD symbol: 1317.
   Margin required for 1.00 Buy Stop order at price 1.31768 on GBPUSD symbol: 1317.68
   Margin required for 1.00 Sell Stop order at price 1.31561 on GBPUSD symbol: 1315.6
   Margin required for 1.00 Buy Stop Limit order at price 1.31718 on GBPUSD symbol: 1
   Margin required for 1.00 Sell Stop Limit order at price 1.31611 on GBPUSD symbol:
   */
  }
//+------------------------------------------------------------------+
//| Return the Buy or Sell order type                                 |
//+------------------------------------------------------------------+
ENUM_ORDER_TYPE MarketOrderByOrderType(ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY  : case ORDER_TYPE_BUY_LIMIT  : case ORDER_TYPE_BUY_STOP  :
        return(ORDER_TYPE_BUY);
      case ORDER_TYPE_SELL : case ORDER_TYPE_SELL_LIMIT : case ORDER_TYPE_SELL_STOP :
        return(ORDER_TYPE_SELL);
     }
   return(WRONG_VALUE);
  }
//+------------------------------------------------------------------+
//| Return open price by order type                                  |
//+------------------------------------------------------------------+
double PriceByOrderType(const string symbol, const ENUM_ORDER_TYPE order_type)
  {
   int     digits=0;
   double  point=0;
   MqlTick tick={};

//--- get the symbol Point value
   ResetLastError();
   if(!SymbolInfoDouble(symbol, SYMBOL_POINT, point))
     {
      Print("SymbolInfoDouble() failed. Error ", GetLastError());
      return 0;
     }

//--- get the symbol Digits value
   long value=0;
   if(!SymbolInfoInteger(symbol, SYMBOL_DIGITS, value))
     {
      Print("SymbolInfoInteger() failed. Error ", GetLastError());
      return 0;
     }
   digits=(int)value;

//--- get the last prices by symbol
   if(!SymbolInfoTick(symbol, tick))
     {
      Print("SymbolInfoTick() failed. Error ", GetLastError());
      return 0;
     }

//--- return the price depending on the order type
   switch(order_type)
     {
      case ORDER_TYPE_BUY              :  return(tick.ask);
      case ORDER_TYPE_SELL             :  return(tick.bid);
      case ORDER_TYPE_BUY_LIMIT        :  return(NormalizeDouble(tick.ask - DEVIATION
      case ORDER_TYPE_SELL_LIMIT       :  return(NormalizeDouble(tick.bid + DEVIATION
      case ORDER_TYPE_BUY_STOP         :  return(NormalizeDouble(tick.ask + DEVIATION
      case ORDER_TYPE_SELL_STOP        :  return(NormalizeDouble(tick.bid - DEVIATION
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return(NormalizeDouble(tick.ask + DEVIATION
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return(NormalizeDouble(tick.bid - DEVIATION
      default                          :  return(0);
     }
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE order_type)
  {
   switch(order_type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      case ORDER_TYPE_CLOSE_BY         :  return("Close By");
      default                          :  return("Unknown order type");
     }
  }
//+------------------------------------------------------------------+
```

See also

OrderSend(), Order Properties, Trade Operation Types

OrderCalcProfit

The function calculates the profit for the current account, in the current market conditions, based on the parameters passed. The function is used for pre-evaluation of the result of a trade operation. The value is returned in the account currency.

```
bool  OrderCalcProfit(
   ENUM_ORDER_TYPE       action,           // type of the order (ORDER_TYPE_BUY or ORD
   string                symbol,           // symbol name
   double                volume,           // volume
   double                price_open,       // open price
   double                price_close,      // close price
   double&               profit            // variable for obtaining the profit value
   );
```

Parameters

action

[in] Type of the order, can be one of the two values of the ENUM_ORDER_TYPE enumeration: ORDER_TYPE_BUY or ORDER_TYPE_SELL.

symbol

[in] Symbol name.

volume

[in] Volume of the trade operation.

price_open

[in] Open price.

price_close

[in] Close price.

profit

[out] The variable, to which the calculated value of the profit will be written in case the function is successfully executed. The estimated profit value depends on many factors, and can differ in different market environments.

Return Value

The function returns true in case of success; otherwise it returns false. If an invalid order type is specified, the function will return false. In order to obtain information about the error, call GetLastError().

Example:

```

#define   VOLUME     1.0   // order volume
#define   DEVIATION  100   // number of points before the close price

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string currency_profit=SymbolInfoString(_Symbol,SYMBOL_CURRENCY_PROFIT);   // profi
   double close_price=ChartPriceOnDropped(); // the price coordinate, corresponding to

//---
   for(int i=0; i<=ORDER_TYPE_SELL; i++)
     {
      ENUM_ORDER_TYPE order_type=(ENUM_ORDER_TYPE)i;                 // order type: 0
      double open_price=PriceOpenByOrderType(_Symbol, order_type);   // open price: fo

      //--- if the open price is not received, continue the loop
      if(open_price==0)
         continue;

      //--- if the close price is zero (the script was not launched by dragging onto t
      if(close_price==0)
         close_price=(order_type==ORDER_TYPE_BUY ? open_price + DEVIATION * _Point : o

      //--- calculate the approximate profit size for the current account and market e
      double profit=0;
      ResetLastError();
      if(!OrderCalcProfit(order_type, _Symbol, VOLUME, open_price, close_price, profit
        {
         Print("OrderCalcProfit() failed. Error ", GetLastError());
         continue;
        }

      //--- print the received profit value in the journal
      PrintFormat("Estimated profit for %.2f %s position on %s with opening price of %
                  VOLUME, OrderTypeDescription(order_type), _Symbol, _Digits, open_pri

     }
   /*
   result:
   Estimated profit for 1.00 Buy position on EURUSD with opening price of 1.10757 and
   Estimated profit for 1.00 Sell position on EURUSD with opening price of 1.10753 and
   */
  }
//+------------------------------------------------------------------+
//| Return open price by order type                                  |
//+------------------------------------------------------------------+
double PriceOpenByOrderType(const string symbol, const ENUM_ORDER_TYPE order_type)
  {
   MqlTick tick={};

//--- get the last prices by symbol
   if(!SymbolInfoTick(symbol, tick))
     {
      Print("SymbolInfoTick() failed. Error ", GetLastError());
      return 0;
     }

//--- return the price depending on the order type
   switch(order_type)
     {
      case ORDER_TYPE_BUY              :  return(tick.ask);
      case ORDER_TYPE_SELL             :  return(tick.bid);
      default                          :  return(0);
     }
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE order_type)
  {
   switch(order_type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      default                          :  return("Unknown order type");
     }
  }
```

See also

OrderSend(), Order Properties, Trade Operation Types

OrderCheck

The OrderCheck() function checks if there are enough money to execute a required trade operation. The check results are placed to the fields of the MqlTradeCheckResult structure.

```
bool  OrderCheck(
   MqlTradeRequest&       request,      // request structure
   MqlTradeCheckResult&   result        // result structure
   );
```

Parameters

request

[in] Pointer to the structure of the MqlTradeRequest type, which describes the required trade action.

result

[in,out] Pointer to the structure of the MqlTradeCheckResult type, to which the check result will be placed.

Return Value

If funds are not enough for the operation, or parameters are filled out incorrectly, the function returns false. In case of a successful basic check of structures (check of pointers), it returns true. **However, this is not an indication that the requested trade operation is sure to be successfully executed**. For a more detailed description of the function execution result, analyze the fields of the *result* structure.

In order to obtain information about the error, call the GetLastError() function.

Example:

```
#define   DEVIATION     5              // allowed deviation from the price
#define   VOLUME        1.0            // order volume
#define   EXPERT_MAGIC  123            // MagicNumber
#define   DIRECTION     ORDER_TYPE_BUY // opened position direction (ORDER_TYPE_BUY or

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the request, verification and result structures
   MqlTradeRequest     request={};
   MqlTradeCheckResult check  ={};
   MqlTradeResult      result ={};

//--- prepare trade request parameters
   PrepareRequest(_Symbol, DIRECTION, VOLUME, request);

//--- check trade request parameters
   ResetLastError();
   bool res=OrderCheck(request, check);
   if(!res)
     {
      PrintFormat("Trade request verification completed with error %d\nServer retcode
      return;
     }

//--- trade request check was successful - display the description of the trade reques
   Print("Trade request verification completed successfully");
   MqlTradeCheckResultPrint(check, 14);

//--- send a trade request
   if(!OrderSend(request, result))
      Print("OrderSend error ", GetLastError());    // if unable to send the request,

//--- information about the operation
   PrintFormat("Trade request result: retcode=%u, deal=%I64u, order=%I64u", result.ret
   /*
   result with disabled auto trading in the client terminal:
   Trade request verification completed with error 4752
   Server retcode: 10027, comment: AutoTrading disabled by client

   enable auto trading and check again on a closed market:
   Experts   automated trading is enabled
   Trade request verification completed successfully
   Retcode:      0
   Balance:      10779.50 USD
   Equity:       10779.50 USD
   Profit:       0.00 USD
   Margin:       1104.79 USD
   Margin free:  9674.71 USD
   Margin level: 975.71 %
   Comment:      Done
   OrderSend error 4756
   Trade request result: retcode=10018, deal=0, order=0

   check on the open market:
   Trade request verification completed successfully
   Retcode:      0
   Balance:      10779.50 USD
   Equity:       10779.50 USD
   Profit:       0.00 USD
   Margin:       110.46 USD
   Margin free:  10669.04 USD
   Margin level: 9758.74 %
   Comment:      Done
   Trade request result: retcode=10009, deal=2777010968, order=2802818813
   */
  }
```

```
//+------------------------------------------------------------------+
//| Prepare parameters for a trade request                           |
//+------------------------------------------------------------------+
void PrepareRequest(const string symbol, const ENUM_ORDER_TYPE order_type, const doubl
  {
   ENUM_ORDER_TYPE type=(DIRECTION !=ORDER_TYPE_BUY ? ORDER_TYPE_SELL : DIRECTION);
   double price=(DIRECTION==ORDER_TYPE_BUY ? SymbolInfoDouble(Symbol(), SYMBOL_ASK) :
//--- request parameters
   request.action    = TRADE_ACTION_DEAL; // trading operation type
   request.symbol    = symbol;            // symbol
   request.volume    = volume;            // volume
   request.type      = type;              // order type
   request.price     = price;             // open price
   request.deviation = DEVIATION;         // allowed deviation from the price
   request.magic     = EXPERT_MAGIC;      // order MagicNumber
  }
//+------------------------------------------------------------------+
//| Print the fields of the trade request                            |
//| verification result in the journal                              |
//+------------------------------------------------------------------+
void MqlTradeCheckResultPrint(const MqlTradeCheckResult &check, const uint header_widt
  {
//--- get the account currency and the number of decimal places for the account curren
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   int    digits  =(int)AccountInfoInteger(ACCOUNT_CURRENCY_DIGITS);

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   string header="Retcode:";
   uint w=(header_width==0 ? header.Length()+1 : header_width);
//--- display the return code with the header of the specified width in the journal
   PrintFormat("%-*s%-u", w, header, check.retcode);

//--- display the balance value after executing a trade operation in the journal
   header="Balance:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.balance, currency);

//--- display the equity value after executing a trade operation in the journal
   header="Equity:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.equity, currency);

//--- display the floating profit value after executing a trading operation in the jou
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.profit, currency);

//--- display the amount of margin, required for the necessary trading operation, in t
```

```
   header="Margin:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.margin, currency);

//--- display the value of equity to be left after conducting a trading operation in t
   header="Margin free:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.*f %s", w, header, digits, check.margin_free, currency);

//--- display the margin level to be set after completing the required trading operat
   header="Margin level:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-.2f %%", w, header, check.margin_level);

//--- display the comment on the response code and error description in the journal
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   PrintFormat("%-*s%-s", w, header, check.comment);
  }
```

See also

OrderSend(), Trade Operation Types, Trade Request Structure, Structure of Request Check Results, Structure of a Trade Request Result

OrderSend

The OrderSend() function is used for executing trade operations by sending requests to a trade server.

```
bool  OrderSend(
   MqlTradeRequest&  request,      // query structure
   MqlTradeResult&   result        // structure of the answer
   );
```

Parameters

request

[in] Pointer to a structure of MqlTradeRequest type describing the trade activity of the client.

result

[in,out] Pointer to a structure of MqlTradeResult type describing the result of trade operation in case of a successful completion (if true is returned).

Return Value

In case of a successful basic check of structures (index checking) returns true. **However, this is not a sign of successful execution of a trade operation**. For a more detailed description of the function execution result, analyze the fields of *result* structure.

Note

The trade requests go through several stages of checking on a trade server. First of all, it checks if all the required fields of the *request* parameter are filled out correctly. If there are no errors, the server accepts the order for further processing. If the order is successfully accepted by the trade server, the OrderSend() function returns true.

It is recommended to check the request before sending it to a trade server. To check requests, use the OrderCheck() function. It checks if there are enough funds to execute the trade operation, and returns many useful parameters in the results of trade request checking:

- return code containing information about errors in the checked request;
- balance value that will appear after the trade operation is executed;
- equity value that will appear after the trade operation is executed;
- floating point value that will appear after the trade operation is executed;
- margin required for the trade operation;
- amount of free equity that will remain after the execution of the trade operation;
- the margin level that will be set after the trade operation is executed;
- comment to the reply code, error description.

When sending a market order (MqlTradeRequest.action=TRADE_ACTION_DEAL), the successful result of the OrderSend() function does not mean that the order has been executed (appropriate trades have been performed). In this case, 'true' means only that the order has been successfully placed in the trading system for further execution. The trade server can fill in the *deal or order* field values *in the returned* result structure, if it is aware of these data when forming a response to an OrderSend() call. Generally, event(s) of executing trades corresponding to an order may happen after sending a response to the OrderSend() call. Therefore, for any type of a trade request, when receiving the OrderSend() execution result, we should first check the *retcode* trade server response code and the *retcode_external* external system response code (if necessary) available in the obtained result structure.

Each accepted order is stored on the trade server awaiting processing until one of the conditions for its execution occurs:

- expiration,
- appearance of an opposite request,
- order execution when the execution price appears,
- a request to cancel the order is received.

At the moment of the order processing, the trade server sends to the terminal a message about the occurrence of the Trade event, which can be processed by the OnTrade() function.

The result of executing the trade request on a server sent by OrderSend() function can be tracked by OnTradeTransaction handler. It should be noted that OnTradeTransaction handler will be called several times when executing one trade request.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. OnTradeTransaction function will be called for each of these events.

Example:

```
//--- value for ORDER_MAGIC
input long order_magic=55555;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- make sure that the account is demo
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("Script operation is not allowed on a live account!");
      return;
     }
//--- place or delete order
   if(GetOrdersTotalByMagic(order_magic)==0)
     {
      //--- no current orders - place an order
      uint res=SendRandomPendingOrder(order_magic);
      Print("Return code of the trade server ",res);
     }
   else // there are orders - delete orders
     {
      DeleteAllOrdersByMagic(order_magic);
     }
//---
  }
//+------------------------------------------------------------------+
//| Receives the current number of orders with specified ORDER_MAGIC |
//+------------------------------------------------------------------+
int GetOrdersTotalByMagic(long const magic_number)
  {
   ulong order_ticket;
   int total=0;
//--- go through all pending orders
   for(int i=0;i<OrdersTotal();i++)
      if((order_ticket=OrderGetTicket(i))>0)
         if(magic_number==OrderGetInteger(ORDER_MAGIC)) total++;
//---
   return(total);
  }
//+------------------------------------------------------------------+
//| Deletes all pending orders with specified ORDER_MAGIC            |
//+------------------------------------------------------------------+
void DeleteAllOrdersByMagic(long const magic_number)
  {
   ulong order_ticket;
//--- go through all pending orders
   for(int i=OrdersTotal()-1;i>=0;i--)
      if((order_ticket=OrderGetTicket(i))>0)
         //--- order with appropriate ORDER_MAGIC
         if(magic_number==OrderGetInteger(ORDER_MAGIC))
           {
            MqlTradeResult result={};
            MqlTradeRequest request={};
            request.order=order_ticket;
            request.action=TRADE_ACTION_REMOVE;
            OrderSend(request,result);
            //--- write the server reply to log
            Print(__FUNCTION__,": ",result.comment," reply code ",result.retcode);
           }
//---
  }
//+------------------------------------------------------------------+
//| Sets a pending order in a random way                             |
//+------------------------------------------------------------------+
uint SendRandomPendingOrder(long const magic_number)
  {
//--- prepare a request
   MqlTradeRequest request={};
   request.action=TRADE_ACTION_PENDING;         // setting a pending order
   request.magic=magic_number;                  // ORDER_MAGIC
   request.symbol=_Symbol;                      // symbol
   request.volume=0.1;                          // volume in 0.1 lots
   request.sl=0;                                // Stop Loss is not specified
   request.tp=0;                                // Take Profit is not specified
//--- form the order type
```

```
   request.type=GetRandomType();                  // order type
//--- form the price for the pending order
   request.price=GetRandomPrice(request.type);  // open price
//--- send a trade request
   MqlTradeResult result={};
   OrderSend(request,result);
//--- write the server reply to log
   Print(__FUNCTION__,":",result.comment);
   if(result.retcode==10016) Print(result.bid,result.ask,result.price);
//--- return code of the trade server reply
   return result.retcode;
  }
//+------------------------------------------------------------------+
//| Returns type of a pending order in a random way                  |
//+------------------------------------------------------------------+
ENUM_ORDER_TYPE GetRandomType()
  {
   int t=MathRand()%4;
//---   0<=t<4
   switch(t)
     {
      case(0):return(ORDER_TYPE_BUY_LIMIT);
      case(1):return(ORDER_TYPE_SELL_LIMIT);
      case(2):return(ORDER_TYPE_BUY_STOP);
      case(3):return(ORDER_TYPE_SELL_STOP);
     }
//--- incorrect value
   return(WRONG_VALUE);
  }
//+------------------------------------------------------------------+
//| Returns price in a random way                                    |
//+------------------------------------------------------------------+
double GetRandomPrice(ENUM_ORDER_TYPE type)
  {
   int t=(int)type;
//--- stop levels for the symbol
   int distance=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
//--- receive data of the last tick
   MqlTick last_tick={};
   SymbolInfoTick(_Symbol,last_tick);
//--- calculate price according to the type
   double price;
   if(t==2 || t==5) // ORDER_TYPE_BUY_LIMIT or ORDER_TYPE_SELL_STOP
     {
      price=last_tick.bid; // depart from price Bid
      price=price-(distance+(MathRand()%10)*5)*_Point;
     }
   else             // ORDER_TYPE_SELL_LIMIT or ORDER_TYPE_BUY_STOP
     {
      price=last_tick.ask; // depart from price Ask
      price=price+(distance+(MathRand()%10)*5)*_Point;
     }
//---
   return(price);
  }
```

See also

Trade Operation Types, Trade Request Structure, Structure of Request Check Results, Structure of a Trade Request Result

OrderSendAsync

The OrderSendAsync() function is used for conducting asynchronous trade operations without waiting for the trade server's response to a sent request. The function is designed for high-frequency trading, when under the terms of the trading algorithm it is unacceptable to waste time waiting for a response from the server.

```
bool  OrderSendAsync(
   MqlTradeRequest&  request,      // Request structure
   MqlTradeResult&   result        // Response structure
   );
```

Parameters

request

[in] A pointer to a structure of the MqlTradeRequest type that describes the trade action of the client.

result

[in,out] A pointer to a structure of the MqlTradeResult type that describes the result of a trade operation in case of successful execution of the function (if true is returned).

Return Value

Returns true if the request is sent to a trade server. In case the request is not sent, it returns false. In case the request is sent, in the *result* variable the response code contains TRADE_RETCODE_PLACED value (code 10008) – "order placed". Successful execution means only the fact of sending, but does not give any guarantee that the request has reached the trade server and has been accepted for processing. When processing the received request, a trade server sends a reply to a client terminal notifying of change in the current state of positions, orders and deals, which leads to the generation of the Trade event.

The result of executing the trade request on a server sent by OrderSendAsync() function can be tracked by OnTradeTransaction handler. It should be noted that OnTradeTransaction handler will be called several times when executing one trade request.

For example, when sending a market buy order, it is handled, an appropriate buy order is created for the account, the order is then executed and removed from the list of the open ones, then it is added to the orders history, an appropriate deal is added to the history and a new position is created. OnTradeTransaction function will be called for each of these events. To get such a data, the function parameters should be analyzed:

- **trans** - this parameter gets MqlTradeTransaction structure describing a trade transaction applied to a trade account;
- **request** - this parameter gets MqlTradeRequest structure describing the trade request resulted in a trade transaction;
- **result** - this parameter gets MqlTradeResult structure describing a trade request execution result.

Note

In terms of purposes and parameters, the function is similar to OrderSend(), but unlike it, it is asynchronous, i.e. does not hold the program operation while waiting for the function execution result. You can compare the rate of trade operations of these two functions using the sample Expert Advisor.

Example:

```
#property description "Expert Advisor for sending trade requests "
                      " using OrderSendAsync() function.\r\n"
#property description "Handling trading events using"
                      " OnTrade() and OnTradeTransaction() handler functions is displa
#property description "Expert Advisor parameters allow setting Magic Number"
                      " (unique ID) "
#property description "and the mode of displaying messages in Experts log. All details
//--- input parameters
input int  MagicNumber=1234567;      // Expert Advisor ID
input bool DescriptionModeFull=true; // Detailed output mode
//--- variable for using in HistorySelect() call
datetime history_start;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- check if autotrading is allowed
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
     {
      Alert("Autotrading in the terminal is disabled, Expert Advisor will be removed."
      ExpertRemove();
      return(-1);
     }
//--- unable to trade on a real account
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("Expert Advisor cannot trade on a real account!");
      ExpertRemove();
      return(-2);
     }
//--- check if it is possible to trade on this account (for example, trading is imposs
   if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
     {
      Alert("Trading on this account is disabled");
      ExpertRemove();
      return(-3);
     }
//--- save the time of launching the Expert Advisor for receiving trading history
   history_start=TimeCurrent();
//---
   CreateBuySellButtons();
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- delete all graphical objects
   ObjectDelete(0,"Buy");
   ObjectDelete(0,"Sell");
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//--- heading named after trading event's handler function
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- receive transaction type as enumeration value
   ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
//--- if transaction is a result of request handling
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      //--- display transaction name
      Print(EnumToString(type));
      //--- then display the string description of the handled request
      Print("------------RequestDescription\r\n",
            RequestDescription(request,DescriptionModeFull));
      //--- and show description of the request result
      Print("------------ ResultDescription\r\n",
            TradeResultDescription(result,DescriptionModeFull));
     }
   else // display full description of the transaction for transactions of another typ
     {
      Print("------------ TransactionDescription\r\n",
            TransactionDescription(trans,DescriptionModeFull));
     }
//---
  }
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- static members for storing trading account status
   static int prev_positions=0,prev_orders=0,prev_deals=0,prev_history_orders=0;
//--- request trading history
   bool update=HistorySelect(history_start,TimeCurrent());
   PrintFormat("HistorySelect(%s , %s) = %s",
               TimeToString(history_start),TimeToString(TimeCurrent()),(string)update)
//--- heading named after trading event's handler function
```

```
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- display handler's name and the number of orders at the moment of handling
   int curr_positions=PositionsTotal();
   int curr_orders=OrdersTotal();
   int curr_deals=HistoryOrdersTotal();
   int curr_history_orders=HistoryDealsTotal();
//--- display the number of orders, positions, deals, as well as changes in parenthese
   PrintFormat("PositionsTotal() = %d (%+d)",
               curr_positions,(curr_positions-prev_positions));
   PrintFormat("OrdersTotal() = %d (%+d)",
               curr_orders,curr_orders-prev_orders);
   PrintFormat("HistoryOrdersTotal() = %d (%+d)",
               curr_deals,curr_deals-prev_deals);
   PrintFormat("HistoryDealsTotal() = %d (%+d)",
               curr_history_orders,curr_history_orders-prev_history_orders);
//--- insert a string break to view the log more conveniently
   Print("");
//--- save the account status
   prev_positions=curr_positions;
   prev_orders=curr_orders;
   prev_deals=curr_deals;
   prev_history_orders=curr_history_orders;
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- handling CHARTEVENT_CLICK event ("Clicking the chart")
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("=> ",__FUNCTION__,": sparam = ",sparam);
      //--- minimum volume for a deal
      double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
      //--- if "Buy" button is pressed, then buy
      if(sparam=="Buy")
        {
         PrintFormat("Buy %s %G lot",_Symbol,volume_min);
         BuyAsync(volume_min);
         //--- unpress the button
         ObjectSetInteger(0,"Buy",OBJPROP_STATE,false);
        }
      //--- if "Sell" button is pressed, then sell
      if(sparam=="Sell")
        {
         PrintFormat("Sell %s %G lot",_Symbol,volume_min);
         SellAsync(volume_min);
         //--- unpress the button
         ObjectSetInteger(0,"Sell",OBJPROP_STATE,false);
        }
      ChartRedraw();
     }
//---
  }
//+------------------------------------------------------------------+
//| Returns the text description of a transaction                    |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans,
                              const bool detailed=true)
  {
//--- prepare a string for returning from the function
   string desc=EnumToString(trans.type)+"\r\n";
//--- all possible data is added in detailed mode
   if(detailed)
     {
      desc+="Symbol: "+trans.symbol+"\r\n";
      desc+="Deal ticket: "+(string)trans.deal+"\r\n";
      desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
      desc+="Order ticket: "+(string)trans.order+"\r\n";
      desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
      desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
      desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
      desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
      desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
      desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
     }
//--- return a received string
   return desc;
  }
//+------------------------------------------------------------------+
//| Returns the text description of the trade request                |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request,
                          const bool detailed=true)
  {
//--- prepare a string for returning from the function
   string desc=EnumToString(request.action)+"\r\n";
//--- add all available data in detailed mode
   if(detailed)
     {
      desc+="Symbol: "+request.symbol+"\r\n";
      desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
      desc+="Order ticket: "+(string)request.order+"\r\n";
      desc+="Order type: "+EnumToString(request.type)+"\r\n";
      desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
      desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
      desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
      desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
      desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
      desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
      desc+="Comment: "+request.comment+"\r\n";
     }
//--- return the received string
   return desc;
  }
//+------------------------------------------------------------------+
//| Returns the text description of request handling result          |
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result,
                              const bool detailed=true)
  {
//--- prepare the string for returning from the function
   string desc="Retcode "+(string)result.retcode+"\r\n";
//--- add all available data in detailed mode
   if(detailed)
     {
      desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
      desc+="Order ticket: "+(string)result.order+"\r\n";
      desc+="Deal ticket: "+(string)result.deal+"\r\n";
      desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
      desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
      desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
      desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
      desc+="Comment: "+result.comment+"\r\n";
     }
//--- return the received string
   return desc;
  }
//+------------------------------------------------------------------+
//| Create two buttons for buying and selling                        |
//+------------------------------------------------------------------+
void CreateBuySellButtons()
  {
//--- check the object named "Buy"
   if(ObjectFind(0,"Buy")>=0)
     {
      //--- if the found object is not a button, delete it
      if(ObjectGetInteger(0,"Buy",OBJPROP_TYPE)!=OBJ_BUTTON)
         ObjectDelete(0,"Buy");
     }
   else
      ObjectCreate(0,"Buy",OBJ_BUTTON,0,0,0); // create "Buy" button
//--- configure "Buy" button
   ObjectSetInteger(0,"Buy",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Buy",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Buy",OBJPROP_YDISTANCE,50);
   ObjectSetInteger(0,"Buy",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Buy",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Buy",OBJPROP_TEXT,"Buy");
   ObjectSetInteger(0,"Buy",OBJPROP_COLOR,clrRed);
//--- check presence of the object named "Sell"
   if(ObjectFind(0,"Sell")>=0)
     {
      //--- if the found object is not a button, delete it
      if(ObjectGetInteger(0,"Sell",OBJPROP_TYPE)!=OBJ_BUTTON)
         ObjectDelete(0,"Sell");
     }
   else
      ObjectCreate(0,"Sell",OBJ_BUTTON,0,0,0); // create "Sell" button
//--- configure "Sell" button
   ObjectSetInteger(0,"Sell",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Sell",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Sell",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Sell",OBJPROP_TEXT,"Sell");
   ObjectSetInteger(0,"Sell",OBJPROP_COLOR,clrBlue);
//--- perform forced update of the chart to see the buttons immediately
   ChartRedraw();
//---
  }
//+------------------------------------------------------------------+
//| Buy using OrderSendAsync() asynchronous function                 |
//+------------------------------------------------------------------+
void BuyAsync(double volume)
  {
//--- prepare the request
   MqlTradeRequest req={};
   req.action      =TRADE_ACTION_DEAL;
   req.symbol      =_Symbol;
   req.magic       =MagicNumber;
   req.volume      =0.1;
   req.type        =ORDER_TYPE_BUY;
   req.price       =SymbolInfoDouble(req.symbol,SYMBOL_ASK);
   req.deviation   =10;
   req.comment     ="Buy using OrderSendAsync()";
   MqlTradeResult  res={};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
//| Sell using OrderSendAsync() asynchronous function                |
//+------------------------------------------------------------------+
void SellAsync(double volume)
  {
//--- prepare the request
   MqlTradeRequest req={};
   req.action      =TRADE_ACTION_DEAL;
   req.symbol      =_Symbol;
   req.magic       =MagicNumber;
   req.volume      =0.1;
   req.type        =ORDER_TYPE_SELL;
   req.price       =SymbolInfoDouble(req.symbol,SYMBOL_BID);
   req.deviation   =10;
   req.comment     ="Sell using OrderSendAsync()";
   MqlTradeResult  res={};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
```

Example of displaying messages in "Experts" log:

```
 12:52:52   ExpertAdvisor (EURUSD,H1)   => OnChartEvent: sparam = Sell
 12:52:52   ExpertAdvisor (EURUSD,H1)   Sell EURUSD 0.01 lot
 12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
 12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_REQUEST
 12:52:52   ExpertAdvisor (EURUSD,H1)   ------------RequestDescription
 12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_ACTION_DEAL
 12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
 12:52:52   ExpertAdvisor (EURUSD,H1)   Magic Number: 1234567
 12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
 12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
 12:52:52   ExpertAdvisor (EURUSD,H1)   Order filling: ORDER_FILLING_FOK
 12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
 12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
 12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
 12:52:52   ExpertAdvisor (EURUSD,H1)   Deviation points: 10
 12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
```

```
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Limit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment: Sell using OrderSendAsync()
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ ResultDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   Retcode 10009
12:52:52   ExpertAdvisor (EURUSD,H1)   Request ID: 2
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Ask: 1.29319
12:52:52   ExpertAdvisor (EURUSD,H1)   Bid: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment:
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+1)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_DELETE
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
```

```
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_HISTORY_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_FILLED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_DEAL_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
```

```
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
```

PositionsTotal

Returns the number of open positions.

```
int  PositionsTotal();
```

Return Value

Value of int type.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get and print the number of open positions on the account in the journal
   int total=PositionsTotal();
   Print("Number of open positions on account: ", total);
   /*
   result:
   Number of open positions on account: 2
   */
  }
```

See also

PositionGetSymbol(), PositionSelect(), Position Properties

PositionGetSymbol

Returns the symbol corresponding to the open position and automatically selects the position for further working with it using functions PositionGetDouble, PositionGetInteger, PositionGetString.

```
string  PositionGetSymbol(
   int   index      // Number in the list of positions
   );
```

Parameters

index

[in] Number of the position in the list of open positions.

Return Value

Value of the string type. If the position was not found, an empty string will be returned. To get an error code, call the GetLastError() function.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the number of open positions on the account
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get position symbol by i loop index
      ResetLastError();
      string symbol=PositionGetSymbol(i);

      //--- if the position symbol is successfully received, then the position at the
      //--- and we can obtain its properties using PositionGetDouble, PositionGetInte
      if(symbol!="")
        {
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE)
         PrintFormat("Position symbol at index %d: %s, position type: %s", i, symbol,
        }
      else
```

```
         {
          PrintFormat("PositionGetSymbol(%d) failed. Error %d", i, GetLastError());
          continue;
         }
      }
    /*
    result:
    Position symbol at index 0: GBPUSD, position type: SELL
    Position symbol at index 1: EURUSD, position type: BUY
    */
  }
```

See also

PositionsTotal(), PositionSelect(), Position Properties

PositionSelect

Chooses an open position for further working with it. Returns true if the function is successfully completed. Returns false in case of failure. To obtain information about the error, call GetLastError().

```
bool  PositionSelect(
   string  symbol      // Symbol name
   );
```

Parameters

symbol

[in] Name of the financial security.

Return Value

Value of the bool type.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol. In this case, PositionSelect will select a position with the lowest ticket.

Function PositionSelect() copies data about a position into the program environment, and further calls of PositionGetDouble(), PositionGetInteger() and PositionGetString() return the earlier copied data. This means that the position itself may no longer exist (or its volume, direction, etc. has changed), but data of this position still can be obtained. To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

Example:

```
#define   SYMBOL_NAME   "EURUSD"

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- select a position on a specified symbol
   if(!PositionSelect(SYMBOL_NAME))
     {
      PrintFormat("PositionSelect(%s) failed. Error %d",SYMBOL_NAME, GetLastError());
      return;
     }

//--- if a position is selected, we can obtain its data using PositionGetDouble(), Pos
//--- get the selected position ticket
   ResetLastError();
   long ticket=PositionGetInteger(POSITION_TICKET);
   if(ticket==0)
     {
      PrintFormat("Failed to get %s position ticket. Error %d", SYMBOL_NAME, GetLastE
      return;
     }

//--- if a ticket is successfully received, print the selected position symbol and tic
   PrintFormat("The position that is selected on the %s symbol has ticket %I64d", SYMB
   /*
   result:
   The position that is selected on the EURUSD symbol has ticket 2810846623
   */
  }
```

See also

PositionGetSymbol(), PositionsTotal(), Position Properties

PositionSelectByTicket

Selects an open position to work with based on the ticket number specified in the position. If successful, returns true. Returns false if the function failed. Call GetLastError() for error details.

```
bool  PositionSelectByTicket(
   ulong    ticket      // Position ticket
   );
```

Parameters

ticket

[in] Position ticket.

Return Value

A value of the bool type.

Note

The PositionSelectByTicket() function copies position data to the program environment. Further calls of PositionGetDouble(), PositionGetInteger() and PositionGetString() return the previously copied data. Even if a position does not exist already (or its size, direction etc. has changed), the data may still be received sometimes. To make sure that you receive valid position data, it is recommended to call PositionSelectByTicket() before you access the data.

Example:

```
#define   EXPERT_MAGIC  123456   // MagicNumber

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare and initialize the request and result structures
   MqlTradeRequest request={};
   MqlTradeResult  result ={};

//--- fill in trade request parameters to open a long position
   request.action    = TRADE_ACTION_DEAL;                     // trading operation ty
   request.symbol    = Symbol();                              // symbol
   request.volume    = 0.1;                                   // volume of 0.1 lot
   request.type      = ORDER_TYPE_BUY;                        // order type
   request.price     = SymbolInfoDouble(Symbol(), SYMBOL_ASK); // open price
   request.deviation = 5;                                     // allowed deviation fi
   request.magic     = EXPERT_MAGIC;                          // order MagicNumber

//--- send a request. If failed to send a request, display the error code and complete
   if(!OrderSend(request, result))
     {
      PrintFormat("OrderSend error ", GetLastError());
      return;
     }

//--- display operation data
   PrintFormat("Trade request result: retcode: %u, deal: %I64u, order: %I64u", result.

//--- get the position ticket from the trade operation result and select the position
//--- the ticket of a newly opened position corresponds to the ticket of the order tha
   ulong ticket=result.order;
   ResetLastError();
   if(!PositionSelectByTicket(ticket))
     {
      PrintFormat("PositionSelectByTicket(%I64u) failed. Error %d", ticket, GetLastEr
      return;
     }

//--- display the data of a position, selected by ticket, in the journal
   ENUM_POSITION_TYPE type  = (ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
   long               time  = PositionGetInteger(POSITION_TIME_MSC);
   double             price = PositionGetDouble(POSITION_PRICE_OPEN);
   double             volume= PositionGetDouble(POSITION_VOLUME);
   string             symbol= PositionGetString(POSITION_SYMBOL);
   int                digits= (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   PrintFormat("Current selected position: %s %.2f %s #%I64u at %.*f, %s",
               symbol, volume, (type==POSITION_TYPE_BUY ? "Buy" : "Sell"), ticket, di
   /*
   result:
   Trade request result: retcode: 10009, deal: 2778100901, order: 2803905975
   Current selected position: EURUSD 0.10 Buy #2803905975 at 1.10672, 2024.09.02 12:0
   */
  }
//+------------------------------------------------------------------+
//| Return time with milliseconds                                    |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000,
  }
```

See also

PositionGetSymbol(), PositionsTotal(), Position Properties

PositionGetDouble

The function returns the requested property of an open position, pre-selected using PositionGetSymbol or PositionSelect. The position property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id      // Property identifier
   );
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variable passed by reference by the last parameter.

```
bool  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id,     // Property identifier
   double&                        double_var       // Here we accept the property valu
   );
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the ENUM_POSITION_PROPERTY_DOUBLE enumeration.

double_var

[out] Variable of the double type, accepting the value of the requested property.

Return Value

Value of the double type. If the function fails, 0 is returned.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- in a loop by all account positions
```

```
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the ticket of the next position by automatically selecting a position
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the position type and display the header for the list of position real
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("Double properties of an open position %s #%I64u:", type, ticket);

      //--- print all the real properties of the selected position under the header
      PositionPropertiesDoublePrint(15);
     }
   /*
   result:
   Double properties of an open position Buy #2807075208:
   Volume:        1.00
   Price open:    1.10516
   StopLoss:      0.00000
   TakeProfit:    0.00000
   Price current: 1.10518
   Swap:          0.00
   Profit:        2.00 USD
   */
  }
//+-------------------------------------------------------------------+
//| Display real properties of the selected position in the journal  |
//+-------------------------------------------------------------------+
void PositionPropertiesDoublePrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;

//--- get the account currency, position symbol and the number of decimal places for t
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
   string symbol  =PositionGetString(POSITION_SYMBOL);
   int    digits  =(int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Volume:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the position volume with the specified header width in the jour
   if(!PositionGetDouble(POSITION_VOLUME, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);
```

```
//--- display the position price value in the journal
   header="Price open:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PRICE_OPEN, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the StopLoss value in the journal
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the TakeProfit value in the journal
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the 'Price current' value in the journal
   header="Price current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PRICE_CURRENT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the accumulated swap value in the journal
   header="Swap:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_SWAP, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the current profit value to the journal
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetDouble(POSITION_PROFIT, value))
      return;
   PrintFormat("%-*s%-.2f %s", w, header, value, currency);
  }
```

See also

PositionGetSymbol(), PositionSelect(), Position Properties

PositionGetInteger

The function returns the requested property of an open position, pre-selected using PositionGetSymbol or PositionSelect. The position property should be of datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id      // Property identifier
   );
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variables passed by reference by the last parameter.

```
bool  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id,     // Property identifier
   long&                           long_var         // Here we accept the property va
   );
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the ENUM_POSITION_PROPERTY_INTEGER enumeration.

long_var

[out] Variable of the long type accepting the value of the requested property.

Return Value

Value of the long type. If the function fails, 0 is returned.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- check if a position is present and display the time of its changing
```

```
   if(PositionSelect(_Symbol))
     {
//--- receive position ID for further work
      ulong position_ID=PositionGetInteger(POSITION_IDENTIFIER);
      Print(_Symbol," position #",position_ID);
//--- receive the time of position forming in milliseconds since 01.01.1970
      long create_time_msc=PositionGetInteger(POSITION_TIME_MSC);
      PrintFormat("Position #%d  POSITION_TIME_MSC = %i64 milliseconds => %s",positio
                  create_time_msc,TimeToString(create_time_msc/1000));
//--- receive the time of the position's last change in seconds since 01.01.1970
      long update_time_sec=PositionGetInteger(POSITION_TIME_UPDATE);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE = %i64 seconds => %s",
                  position_ID,update_time_sec,TimeToString(update_time_sec));
//--- receive the time of the position's last change in milliseconds since 01.01.1970
      long update_time_msc=PositionGetInteger(POSITION_TIME_UPDATE_MSC);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE_MSC = %i64 milliseconds => %s",
                  position_ID,update_time_msc,TimeToString(update_time_msc/1000));
     }
//---
  }
```

See also

PositionGetSymbol(), PositionSelect(), Position Properties

PositionGetString

The function returns the requested property of an open position, pre-selected using PositionGetSymbol or PositionSelect. The position property should be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id      // Property identifier
   );
```

2. Returns true or false, depending on the success of the function execution. If successful, the value of the property is placed in a receiving variables passed by reference by the last parameter.

```
bool  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id,     // Property identifier
   string&                        string_var       // Here we accept the property valu
   );
```

Parameters

property_id

[in] Identifier of a position property. The value can be one of the values of the ENUM_POSITION_PROPERTY_STRING enumeration.

string_var

[out] Variable of the string type accepting the value of the requested property.

Return Value

Value of the string type. If the function fails, an empty string is returned.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- in a loop by all account positions
```

```
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the ticket of the next position by automatically selecting a position
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the position type and display the header for the list of position str
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("String properties of an open position %s #%I64u:", type, ticket);

      //--- print all the string properties of the selected position under the header
      PositionPropertiesStringPrint(15);
     }
   /*
   result:
   String properties of an open position Buy #2810798881:
   Symbol:        EURUSD
   Comment:       Test PositionGetString
   External ID:
   */
  }
//+-------------------------------------------------------------------+
//| Display string properties of the selected position in the journal|
//+-------------------------------------------------------------------+
void PositionPropertiesStringPrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value="";

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the position symbol with the specified header width in the jour
   if(!PositionGetString(POSITION_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- display the position comment in the journal
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetString(POSITION_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- display the position ID in an external system in the journal
```

```
   header="External ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!PositionGetString(POSITION_EXTERNAL_ID, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
  }
```

See also

PositionGetSymbol(), PositionSelect(), Position Properties

PositionGetTicket

The function returns the ticket of a position with the specified index in the list of open positions and automatically selects the position to work with using functions PositionGetDouble, PositionGetInteger, PositionGetString.

```
ulong  PositionGetTicket(
   int  index      // The number of a position in the list
   );
```

Parameters

index

[in] The index of a position in the list of open positions, numeration starts with 0.

Return Value

The ticket of the position. Returns 0 if the function fails.

Note

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about a position, it is recommended to call PositionSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- in a loop by all account positions
   int total=PositionsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the ticket of the next position by automatically selecting a position
      ulong ticket=PositionGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the position type and display the description of the selected positio
      string type=(PositionGetInteger(POSITION_TYPE)==POSITION_TYPE ? "Buy" : "Sell");
      PrintFormat("[%d] Selected position %s #%I64u", i, type, ticket);
     }
   /*
```

```
    result:
    [0] Selected position Sell #2810802718
    [1] Selected position Buy #2810802919
    */
   }
```

See also

PositionGetSymbol(), PositionSelect(), Position Properties

OrdersTotal

Returns the number of current orders.

```
int  OrdersTotal();
```

Return Value

Value of the int type.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal. An order is a request to conduct a transaction, while a position is a result of one or more deals.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get and print in the journal the number of active pending orders on the account
   int total=OrdersTotal();
   Print("Number of active pending orders on the account: ", total);
   /*
   result:
   Number of active pending orders on the account: 2
   */
  }
```

See also

OrderSelect(), OrderGetTicket(), Order Properties

OrderGetTicket

Returns ticket of a corresponding order and automatically selects the order for further working with it using functions.

```
ulong  OrderGetTicket(
   int  index      // Number in the list of orders
   );
```

Parameters

index

[in] Number of an order in the list of current orders.

Return Value

Value of the ulong type. If the function fails, 0 is returned.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal. An order is a request to conduct a transaction, while a position is a result of one or more deals.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

Function OrderGetTicket() copies data about an order into the program environment, and further calls of OrderGetDouble(), OrderGetInteger(), OrderGetString() return the earlier copied data. This means that the order itself may no longer exist (or its open price, Stop Loss/Take Profit levels or expiration has changed), but data of this order still can be obtained. To ensure receipt of fresh data about an order, it is recommended to call OrderGetTicket() right before referring to them.

Example:

```
void OnStart()
  {
//--- variables for returning values from order properties
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   string   symbol;
   string   type;
   long     order_magic;
   long     positionID;
//--- number of current pending orders
   uint     total=OrdersTotal();

//--- go through orders in a loop
   for(uint i=0;i<total;i++)
     {
      //--- return order ticket by its position in the list
      if((ticket=OrderGetTicket(i))>0)
        {
         //--- return order properties
         open_price    =OrderGetDouble(ORDER_PRICE_OPEN);
         time_setup    =(datetime)OrderGetInteger(ORDER_TIME_SETUP);
         symbol        =OrderGetString(ORDER_SYMBOL);
         order_magic   =OrderGetInteger(ORDER_MAGIC);
         positionID    =OrderGetInteger(ORDER_POSITION_ID);
         initial_volume=OrderGetDouble(ORDER_VOLUME_INITIAL);
         type          =EnumToString(ENUM_ORDER_TYPE(OrderGetInteger(ORDER_TYPE)));
         //--- prepare and show information about the order
         printf("#ticket %d %s %G %s at %G was set up at %s",
                ticket,                 // order ticket
                type,                   // type
                initial_volume,         // placed volume
                symbol,                 // symbol
                open_price,             // specified open price
                TimeToString(time_setup)// time of order placing
                );
        }
     }
//---
  }
```

See also

OrdersTotal(), OrderSelect(), OrderGetInteger()

OrderSelect

Selects an order to work with. Returns true if the function has been successfully completed. Returns false if the function completion has failed. For more information about an error call GetLastError().

```
bool  OrderSelect(
   ulong   ticket      // Order ticket
   );
```

Parameters

ticket

[in] Order ticket.

Return Value

Value of the bool type.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

Function OrderSelect() copies data about an order into the program environment, and further calls of OrderGetDouble(), OrderGetInteger(), OrderGetString() return the earlier copied data. This means that the order itself may no longer exist (or its open price, Stop Loss/Take Profit levels or expiration has changed), but data of this order still can be obtained. To ensure receipt of fresh data about an order, it is recommended to call OrderSelect() right before referring to them.

Example:

```
#define   EXPERT_MAGIC  123456
#define   OFFSET        50                       // offset from the current price to pl
#define   DIRECTION     ORDER_TYPE_BUY_LIMIT     // order type
#define   VOLUME        1.0                      // volume
#define   DEVIATION     2                        // allowed deviation from the price

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//-- declare and initialize the trade request, result and variables
   MqlTradeRequest request={};
   MqlTradeResult  result ={};
   double          order_price=0;

//--- pending order placing parameters
   request.action     = TRADE_ACTION_PENDING;                               // trading
   request.symbol     = _Symbol;                                            // symbol
   request.volume     = VOLUME;                                             // volume
   request.deviation = DEVIATION;                                           // allowed
   request.magic      = EXPERT_MAGIC;                                       // order Ma

//--- check operation type
   switch(DIRECTION)
     {
      case ORDER_TYPE_BUY_LIMIT :
        request.type = ORDER_TYPE_BUY_LIMIT;                               // order ty
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_ASK)-OFFSET*_Point;// open pri
        request.price= NormalizeDouble(order_price, _Digits);              // normaliz
        break;
      case ORDER_TYPE_SELL_LIMIT :
        request.type = ORDER_TYPE_SELL_LIMIT;                              // order ty
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_BID)+OFFSET*_Point;// open pri
        request.price= NormalizeDouble(order_price,_Digits);               // normaliz
        break;
      case ORDER_TYPE_BUY_STOP :
        request.type = ORDER_TYPE_BUY_STOP;                                // order ty
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_ASK)+OFFSET*_Point;// open pri
        request.price= NormalizeDouble(order_price,_Digits);               // normaliz
        break;
      case ORDER_TYPE_SELL_STOP :
        request.type = ORDER_TYPE_SELL_STOP;                               // order ty
        order_price  = SymbolInfoDouble(_Symbol, SYMBOL_BID)-OFFSET*_Point;// open pri
        request.price= NormalizeDouble(order_price,_Digits);               // normaliz
        break;
      default: // if non-pending or StopLimit order is selected
        Alert("This example is only for placing pending orders BuyLimit, SellLimit, Bu
        break;
     }

//--- send a request. If failed to send a request, display the error code and complete
   if(!OrderSend(request, result))
     {
      Print("OrderSend error ", GetLastError());
      return;
     }

//--- display operation data
   PrintFormat("Trade request result: retcode=%u, order=%I64u", result.retcode, result

//--- get the order ticket from the trade operation result and select the order by ti
   ulong ticket=result.order;
   ResetLastError();
   if(!OrderSelect(ticket))
     {
      PrintFormat("OrderSelect(%I64u) failed. Error %d", ticket, GetLastError());
      return;
     }

//--- display the data of the order, selected by ticket, in the journal
   ENUM_ORDER_TYPE   type  = (ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE);
   long              time  = OrderGetInteger(ORDER_TIME_SETUP_MSC);
   double            price = OrderGetDouble(ORDER_PRICE_OPEN);
   double            volume= OrderGetDouble(ORDER_VOLUME_CURRENT);
   string            symbol= OrderGetString(ORDER_SYMBOL);
   int               digits= (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   PrintFormat("Current selected order: %s %.2f %s #%I64u at %.*f, %s",
               symbol, volume, OrderTypeDescription(type), ticket, digits, price, Time
   /*
   result:
   Trade request result: retcode=10009, order=2811006719
   Current selected order: EURUSD 1.00 Buy Limit #2811006719 at 1.10550, 2024.09.04 1
   */
  }
//+------------------------------------------------------------------+
//| Return time with milliseconds                                    |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
```

See also

OrderGetInteger(), OrderGetDouble(), OrderGetString(), OrderCalcProfit(), OrderGetTicket(), Order Properties

OrderGetDouble

Returns the requested property of an order, pre-selected using OrderGetTicket or OrderSelect. The order property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id        // Property identifier
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable passed by reference by the last parameter.

```
bool  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id,        // Property identifier
   double&                     double_var          // Here we accept the property va
   );
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_DOUBLE enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the double type. If the function fails, 0 is returned.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call OrderSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- in a loop by the list of all account orders
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the order ticket in the list by the loop index
      ulong ticket=OrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the order type and display the header for the list of real properties
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
      PrintFormat("Double properties of an active pending order %s #%I64u:", type, ti

      //--- print all the real properties of the selected order under the header
      OrderPropertiesDoublePrint(16);
     }
   /*
   result:
   Double properties of an active pending order Sell Limit #2812000714:
   Volume initial: 1.00
   Volume current: 1.00
   Price open:     145.282
   StopLoss:       0.000
   TakeProfit:     0.000
   Price current:  145.044
   StopLimit:      0.000
   Double properties of an active pending order Buy Limit #2812001112:
   Volume initial: 1.00
   Volume current: 1.00
   Price open:     144.836
   StopLoss:       0.000
   TakeProfit:     0.000
   Price current:  145.051
   StopLimit:      0.000
   Double properties of an active pending order Buy Stop #2812001488:
   Volume initial: 0.50
   Volume current: 0.50
   Price open:     1.10642
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10530
   StopLimit:      0.00000
   Double properties of an active pending order Sell Stop #2812001712:
   Volume initial: 0.50
   Volume current: 0.50
   Price open:     1.10374
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10525
   StopLimit:      0.00000
   */
  }
//+------------------------------------------------------------------+
//| Return the order type description                               |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
//+------------------------------------------------------------------+
//| Display real properties of the selected order in the journal     |
//+------------------------------------------------------------------+
void OrderPropertiesDoublePrint(const uint header_width=0)
  {
//--- get the order symbol and the number of decimal places for the symbol
   string symbol = OrderGetString(ORDER_SYMBOL);
   int    digits = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- display the initial volume when placing an order with a header of the specified
   OrderPropertyPrint("Volume initial:",header_width,2,ORDER_VOLUME_INITIAL);

//--- display the unfulfilled order volume in the journal
   OrderPropertyPrint("Volume current:",header_width,2,ORDER_VOLUME_CURRENT);

//--- display the price, specified in the order, in the journal
   OrderPropertyPrint("Price open:",header_width,digits,ORDER_PRICE_OPEN);

//--- display the StopLoss level in the journal
   OrderPropertyPrint("StopLoss:",header_width,digits,ORDER_SL);

//--- display the TakeProfit level in the journal
   OrderPropertyPrint("TakeProfit:",header_width,digits,ORDER_TP);

//--- display the current price by the order symbol in the journal
   OrderPropertyPrint("Price current:",header_width,digits,ORDER_PRICE_CURRENT);

//--- display the Limit order price, when StopLimit order is activated, in the journa
```

```
    OrderPropertyPrint("StopLimit:",header_width,digits,ORDER_PRICE_STOPLIMIT);
   }
//+------------------------------------------------------------------+
//| Display the order real property value in the journal             |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, int digits, ENUM_ORDEI
  {
   double value=0;
   if(!OrderGetDouble(property, value))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLastE
   else
     {
      //--- if the header width is passed to the function equal to zero, then the widt
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-.*f", w, header, digits, value);
     }
  }
```

See also

OrdersTotal(), OrderGetTicket(), Order Properties

OrderGetInteger

Returns the requested order property, pre-selected using OrderGetTicket or OrderSelect. Order property must be of the datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id        // Property identifier
   );
```

2. Returns true or false depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id,       // Property identifier
   long&                        long_var           // Here we accept the property valu
   );
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_INTEGER enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the long type. If the function fails, 0 is returned.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call OrderSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- in a loop by the list of all account orders
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the order ticket in the list by the loop index
      ulong ticket=OrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the order type and display the header for the list of real properties
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
      PrintFormat("Integer properties of an active pending order %s #%I64u:", type, t

      //--- print all the integer properties of the selected order under the header
      OrderPropertiesIntegerPrint(17);
     }
   /*
   result:
   Integer properties of an active pending order Buy Limit #2812945317:
   Ticket:          2812945317
   Time setup:      2024.09.04 19:17:16
   Type:            Buy Limit
   State:           Placed
   Time expiration: 0
   Time done:       0
   Time setup msc:  2024.09.04 19:17:16.686
   Time done msc:   0
   Type filling:    Return
   Type time:       Time GTC
   Magic:           0
   Reason:          Client
   Position ID:     0
   Position By ID:  0
   */
  }
//+------------------------------------------------------------------+
//| Display integer properties of the selected order in the journal  |
//+------------------------------------------------------------------+
void OrderPropertiesIntegerPrint(const uint header_width=0)
  {
   uint   w=0;
   string header="";
   long   value=0;

//--- display order ticket in the journal
   OrderPropertyPrint("Ticket:", header_width, ORDER_TICKET);

//--- display the order placement time in the journal
   OrderPropertyPrint("Time setup:", header_width, ORDER_TIME_SETUP);
```

```
//--- display the order type in the journal
   OrderPropertyPrint("Type:", header_width, ORDER_TYPE);

//--- display the order status in the journal
   OrderPropertyPrint("State:", header_width, ORDER_STATE);

//--- display the order expiration time in the journal
   OrderPropertyPrint("Time expiration:", header_width, ORDER_TIME_EXPIRATION);

//--- display the order execution/expiration time in the journal
   OrderPropertyPrint("Time done:", header_width, ORDER_TIME_DONE);

//--- display the time of placing an order for execution in milliseconds since 01.01.
   OrderPropertyPrint("Time setup msc:", header_width, ORDER_TIME_SETUP_MSC);

//--- display the order execution/expiration time in milliseconds since 01.01.1970 in
   OrderPropertyPrint("Time done msc:", header_width, ORDER_TIME_DONE_MSC);

//--- display the execution type by residue in the journal
   OrderPropertyPrint("Type filling:", header_width, ORDER_TYPE_FILLING);

//--- display the order lifetime in the journal
   OrderPropertyPrint("Type time:", header_width, ORDER_TYPE_TIME);

//--- display the ID of the EA, that placed an order, in the journal
   OrderPropertyPrint("Magic:", header_width, ORDER_MAGIC);

//--- display order reason or source in the journal
   OrderPropertyPrint("Reason:", header_width, ORDER_REASON);

//--- display the ID of the position, set on the order during its execution, in the j
   OrderPropertyPrint("Position ID:", header_width, ORDER_POSITION_ID);

//--- display the opposite position ID for ORDER_TYPE_CLOSE_BY type orders in the jou
   OrderPropertyPrint("Position By ID:", header_width, ORDER_POSITION_BY_ID);
  }
//+------------------------------------------------------------------+
//| Display the order integer property value in the journal          |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, ENUM_ORDER_PROPERTY_I
  {
   string svalue="";
   long   lvalue=0;
   if(!OrderGetInteger(property, lvalue))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLast
   else
     {
      switch(property)
```

```
        {
         case ORDER_TICKET          :
         case ORDER_MAGIC           :
         case ORDER_POSITION_ID     :
         case ORDER_POSITION_BY_ID  :
           svalue=(string)lvalue;
           break;

         case ORDER_TIME_SETUP      :
         case ORDER_TIME_EXPIRATION :
         case ORDER_TIME_DONE       :
           svalue=(lvalue!=0 ? TimeToString((datetime)lvalue, TIME_DATE|TIME_MINUTES|
           break;

         case ORDER_TIME_SETUP_MSC  :
         case ORDER_TIME_DONE_MSC   :
           svalue=(lvalue!=0 ? TimeMscToString(lvalue) : "0");
           break;

         case ORDER_TYPE            :
           svalue=OrderTypeDescription((ENUM_ORDER_TYPE)lvalue);
           break;
         case ORDER_STATE           :
           svalue=OrderStateDescription((ENUM_ORDER_STATE)lvalue);
           break;
         case ORDER_TYPE_FILLING    :
           svalue=OrderTypeFillingDescription((ENUM_ORDER_TYPE_FILLING)lvalue);
           break;
         case ORDER_TYPE_TIME       :
           svalue=OrderTypeTimeDescription((ENUM_ORDER_TYPE_TIME)lvalue);
           break;
         case ORDER_REASON          :
           svalue=OrderReasonDescription((ENUM_ORDER_REASON)lvalue);
           break;

         default                    :
           svalue="Unknown property";
           break;
        }

      //--- if the header width is passed to the function equal to zero, then the widt
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-s", w, header, svalue);
     }
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
```

```
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return the order status description                              |
//+------------------------------------------------------------------+
string OrderStateDescription(ENUM_ORDER_STATE state)
  {
   switch(state)
     {
      case ORDER_STATE_STARTED         :  return("Started");
      case ORDER_STATE_PLACED          :  return("Placed");
      case ORDER_STATE_CANCELED        :  return("Canceled");
      case ORDER_STATE_PARTIAL         :  return("Partial");
      case ORDER_STATE_FILLED          :  return("Filled");
      case ORDER_STATE_REJECTED        :  return("Rejected");
      case ORDER_STATE_EXPIRED         :  return("Expired");
      case ORDER_STATE_REQUEST_ADD     :  return("Request Add");
      case ORDER_STATE_REQUEST_MODIFY  :  return("Request Modify");
      case ORDER_STATE_REQUEST_CANCEL  :  return("Request Cancel");
      default                          :  return("Unknown state: "+(string)state);
     }
  }
//+------------------------------------------------------------------+
//| Return the description of the order volume filling policy        |
//+------------------------------------------------------------------+
string OrderTypeFillingDescription(const ENUM_ORDER_TYPE_FILLING type)
  {
   switch(type)
     {
      case ORDER_FILLING_FOK     :  return("Fill or Kill");
      case ORDER_FILLING_IOC     :  return("Immediate or Cancel");
      case ORDER_FILLING_BOC     :  return("Book or Cancel");
      case ORDER_FILLING_RETURN  :  return("Return");
      default                    :  return("Unknown type filling: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return the order expiration date description                     |
//+------------------------------------------------------------------+
string OrderTypeTimeDescription(const ENUM_ORDER_TYPE_TIME type)
  {
   switch(type)
     {
      case ORDER_TIME_GTC           :   return("Time GTC");
      case ORDER_TIME_DAY           :   return("Time Day");
      case ORDER_TIME_SPECIFIED     :   return("Time Specified");
      case ORDER_TIME_SPECIFIED_DAY :   return("Time Specified Day");
      default                       :  return("Unknown type time: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return the order placement reason description                    |
//+------------------------------------------------------------------+
string OrderReasonDescription(const ENUM_ORDER_REASON reason)
  {
   switch(reason)
     {
      case ORDER_REASON_CLIENT   :  return("Client");
      case ORDER_REASON_MOBILE   :  return("Mobile");
      case ORDER_REASON_WEB      :  return("Web");
      case ORDER_REASON_EXPERT   :  return("Expert");
      case ORDER_REASON_SL       :  return("Stop Loss");
      case ORDER_REASON_TP       :  return("Take Profit");
      case ORDER_REASON_SO       :  return("Stop Out");
      default                    :  return("Unknown reason: "+(string)reason);
     }
  }
//+------------------------------------------------------------------+
//| Return time with milliseconds                                    |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
```

See also

[OrdersTotal()](), [OrderGetTicket()](), [Order Properties]()

OrderGetString

Returns the requested order property, pre-selected using OrderGetTicket or OrderSelect. The order property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id        // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id,       // Property identifier
   string&                     string_var         // Here we accept the property valu
   );
```

Parameters

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_STRING enumeration.

string_var

[out] Variable of the string type that accepts the value of the requested property.

Return Value

Value of the string type.

Note

Do not confuse current pending orders with positions, which are also displayed on the "Trade" tab of the "Toolbox" of the client terminal.

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol.

To ensure receipt of fresh data about an order, it is recommended to call OrderSelect() right before referring to them.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- in a loop by the list of all account orders
   int total=OrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the order ticket in the list by the loop index
      ulong ticket=OrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the order type and display the header for the list of string propertie
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE));
      PrintFormat("String properties of an active pending order %s #%I64u:", type, tic

      //--- print all the string properties of the selected order under the header
      OrderPropertiesStringPrint(13);
     }
   /*
   result:
   String properties of an active pending order Sell Limit #2813781342:
   Comment:     Test OrderGetString
   Symbol:      EURUSD
   External ID:
   */
  }
//+------------------------------------------------------------------+
//| Display string properties of the selected order in the journal   |
//+------------------------------------------------------------------+
void OrderPropertiesStringPrint(const uint header_width=0)
  {
//--- display the comment in the journal
   OrderPropertyPrint("Comment:", header_width, ORDER_COMMENT);

//--- display a symbol the order has been placed for in the journal
   OrderPropertyPrint("Symbol:", header_width, ORDER_SYMBOL);

//--- display the order ID in an external system in the journal
   OrderPropertyPrint("External ID:", header_width, ORDER_EXTERNAL_ID);
  }
//+------------------------------------------------------------------+
//| Display the order string property value in the journal           |
//+------------------------------------------------------------------+
void OrderPropertyPrint(const string header, uint header_width, ENUM_ORDER_PROPERTY_ST
  {
   string value="";
   if(!OrderGetString(property, value))
      PrintFormat("Cannot get property %s, error=%d", EnumToString(property), GetLastE
   else
     {
      //--- if the header width is passed to the function equal to zero, then the widt
      uint w=(header_width==0 ? header.Length()+1 : header_width);
      PrintFormat("%-*s%-s", w, header, value);
     }
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
```

See also

OrdersTotal(), OrderGetTicket(), Order Properties

HistorySelect

Retrieves the history of deals and orders for the specified period of server time.

```
bool  HistorySelect(
   datetime  from_date,      // From date
   datetime  to_date         // To date
   );
```

Parameters

from_date

[in] Start date of the request.

to_date

[in] End date of the request.

Return Value

It returns true if successful, otherwise returns false.

Note

HistorySelect() creates a list of orders and a list of trades in a mql5-program, for further referring to the list elements using corresponding functions. The deals list size can be returned using the HistoryDealsTotal() function; the size of the list of orders in the history can be obtained using HistoryOrdersTotal(). Selection in the list of orders should be better performed by HistoryOrderGetTicket(), for items in the list of deals HistoryDealGetTicket() suits better.

After using HistoryOrderSelect(), the list of history orders available to the mql5 program is reset and filled again by the found order, if the search of an order by the ticket has been completed successfully. The same applies to the list of deals available to the mql5 program - it is reset by HistoryDealSelect() and filled again in case of a successful receipt of a deal by ticket number.

Example:

```
void OnStart()
  {
   color BuyColor =clrBlue;
   color SellColor=clrRed;
//--- request trade history
   HistorySelect(0,TimeCurrent());
//--- create objects
   string   name;
   uint     total=HistoryDealsTotal();
   ulong    ticket=0;
   double   price;
   double   profit;
   datetime time;
   string   symbol;
   long     type;
   long     entry;
//--- for all deals
```

```
   for(uint i=0;i<total;i++)
     {
      //--- try to get deals ticket
      if((ticket=HistoryDealGetTicket(i))>0)
        {
         //--- get deals properties
         price =HistoryDealGetDouble(ticket,DEAL_PRICE);
         time  =(datetime)HistoryDealGetInteger(ticket,DEAL_TIME);
         symbol=HistoryDealGetString(ticket,DEAL_SYMBOL);
         type  =HistoryDealGetInteger(ticket,DEAL_TYPE);
         entry =HistoryDealGetInteger(ticket,DEAL_ENTRY);
         profit=HistoryDealGetDouble(ticket,DEAL_PROFIT);
         //--- only for current symbol
         if(price && time && symbol==Symbol())
           {
            //--- create price object
            name="TradeHistory_Deal_"+string(ticket);
            if(entry) ObjectCreate(0,name,OBJ_ARROW_RIGHT_PRICE,0,time,price,0,0);
            else      ObjectCreate(0,name,OBJ_ARROW_LEFT_PRICE,0,time,price,0,0);
            //--- set object properties
            ObjectSetInteger(0,name,OBJPROP_SELECTABLE,0);
            ObjectSetInteger(0,name,OBJPROP_BACK,0);
            ObjectSetInteger(0,name,OBJPROP_COLOR,type?BuyColor:SellColor);
            if(profit!=0) ObjectSetString(0,name,OBJPROP_TEXT,"Profit: "+string(profit
           }
        }
     }
//--- apply on chart
   ChartRedraw();
  }
```

See also

[HistoryOrderSelect()](), [HistoryDealSelect()]()

HistorySelectByPosition

Retrieves the history of deals and orders having the specified position identifier.

```
bool  HistorySelectByPosition(
   ulong    position_id     // position identifier - POSITION_IDENTIFIER
   );
```

Parameters

position_id

[in] Position identifier that is set to every executed order and every deal.

Return Value

It returns true if successful, otherwise returns false.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

HistorySelectByPosition() creates in a mql5 program a list of orders and a list of deals with a specified position identifier for further reference to the elements of the list using the appropriate functions. To know the size of the list of deals, use function HistoryDealsTotal(), the size of the list of orders in the history can be obtained using HistoryOrdersTotal(). To run through elements of the orders list, use HistoryOrderGetTicket(), for elements of the deals list - HistoryDealGetTicket().

After using HistoryOrderSelect(), list of history orders available to the mql5 program is reset and filled again with the found order, if search of an order by its ticket was successful. The same refers to the list of deals available to the mql5 program - it is reset by function HistoryDealSelect() and is filled out again if a deal was found successfully by the ticket number.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   long  pos_id_array[];         // array for storing position IDs

//--- request the entire history
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- collect all Position IDs from pending orders only in the array
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
```

```
     {
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;
      ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket, ORDER_TYPE)
      long pos_id=HistoryOrderGetInteger(ticket, ORDER_POSITION_ID);
      if(type<=ORDER_TYPE_SELL || pos_id==0)
         continue;

      int size=ArraySize(pos_id_array);
      if(ArrayResize(pos_id_array, size+1)==size+1)
         pos_id_array[size]=pos_id;
     }

//--- by list of position IDs in the array
   total=ArraySize(pos_id_array);
   for(int i=0; i<total; i++)
     {
      //--- print the header, as well as the position order and deal list
      long position_id=pos_id_array[i];
      Print("List of orders and deals for position with ID: ", position_id);
      HistorySelectByPositionProcess(position_id);
     }
   /*
   result:
   List of orders and deals for position with ID: 1819629924
     [0] Order Sell Limit #1819629924
     [1] Order Buy #1819633194
     [0] Entry In Deal Sell #1794972472
     [1] Entry Out Deal Buy #1794975589
   List of orders and deals for position with ID: 1841753970
     [0] Order Sell Stop #1841753970
     [1] Order Buy #1842322160
     [0] Entry In Deal Sell #1817242142
     [1] Entry Out Deal Buy #1817765341
   */
  }
//+------------------------------------------------------------------+
//| Select history of orders and deals by position ID and            |
//| prints a list of orders and deals for the position in the journal|
//+------------------------------------------------------------------+
bool HistorySelectByPositionProcess(const long position_id)
  {
//--- request the history of deals and orders having the specified position ID
   if(!HistorySelectByPosition(position_id))
     {
      PrintFormat("HistorySelectByPosition(%I64d) failed. Error %d", position_id, GetL
      return(false);
     }
```

```
//--- print a list of position orders
   int orders_total=HistoryOrdersTotal();
   for(int i=0; i<orders_total; i++)
     {
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;
      ENUM_ORDER_TYPE order_type=(ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket, ORDE
      PrintFormat("  [%d] Order %s #%I64u", i, OrderTypeDescription(order_type), ticke
     }

//--- print a list of position deals in the journal
   int deals_total =HistoryDealsTotal();
   for(int i=0; i<deals_total; i++)
     {
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;
      ENUM_DEAL_ENTRY deal_entry=(ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket, DEAL_
      ENUM_DEAL_TYPE  deal_type= (ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, DEAL_T
      if(deal_type!=DEAL_TYPE_BUY && deal_type!=DEAL_TYPE_SELL)
         continue;
      PrintFormat("  [%d] Entry %s Deal %s #%I64u", i, DealEntryDescription(deal_entr
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return the position deal type description                        |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
```

```
  {
   switch(type)
     {
      //--- return the description of the Buy and Sell deals only,
      //--- since all other types do not apply to the position
      case DEAL_TYPE_BUY    :  return("Buy");
      case DEAL_TYPE_SELL   :  return("Sell");
      default               :  return("Unknown deal type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return position change method                                    |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN       :  return("In");
      case DEAL_ENTRY_OUT      :  return("Out");
      case DEAL_ENTRY_INOUT    :  return("InOut");
      case DEAL_ENTRY_OUT_BY   :  return("Out by");
      case DEAL_ENTRY_STATE    :  return("Status record");
      default                  :  return("Unknown deal entry: "+(string)entry);
     }
  }
```

See also

HistorySelect(), HistoryOrderGetTicket(), Order Properties

HistoryOrderSelect

Selects an order from the history for further calling it through appropriate functions. It returns true if the function has been successfully completed. Returns false if the function has failed. For more details on error call GetLastError().

```
bool  HistoryOrderSelect(
   ulong   ticket      // Order ticket
   );
```

Parameters

ticket

[in] Order ticket.

Return Value

Returns true if successful, otherwise false.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

HistoryOrderSelect() clears in a mql5-program the list of orders from a history, available for calls, and copies to it a single order, if the execution of HistoryOrderSelect () has been completed successfully. If you need to go through all orders selected by HistorySelect(), you should better use HistoryOrderGetTicket().

Example:

```
#define   TICKET    1819621374   // ticket of any known order, for example, from the
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- select a historical order by the ticket specified in TICKET
   if(!HistoryOrderSelect(TICKET))
     {
      PrintFormat("HistoryOrderSelect(%I64u) failed. Error %d", TICKET, GetLastError()
      return;
     }
     
//--- if an order is successfully selected, get its data and display the order descri
   ENUM_ORDER_TYPE   order_type  = (ENUM_ORDER_TYPE)HistoryOrderGetInteger(TICKET, OR
   ENUM_ORDER_STATE  order_state = (ENUM_ORDER_STATE)HistoryOrderGetInteger(TICKET, O
   ENUM_ORDER_REASON order_reason= (ENUM_ORDER_REASON)HistoryOrderGetInteger(TICKET, 
   long              order_time  = HistoryOrderGetInteger(TICKET, ORDER_TIME_SETUP_MS
   string            order_symbol=HistoryOrderGetString(TICKET, ORDER_SYMBOL);
   double            order_vol_init=HistoryOrderGetDouble(TICKET, ORDER_VOLUME_INITIA
   double               order_vol_curr=HistoryOrderGetDouble(TICKET, ORDER_VOLUME_CURRENT
   PrintFormat("%s Order %.2f/%s %s #%I64u %s by %s at %s",
               order_symbol, order_vol_init, (order_vol_curr>0 ? DoubleToString(order_
               OrderTypeDescription(order_type), TICKET, OrderStateDescription(order_
               OrderReasonDescription(order_reason), TimeMscToString(order_time));
   /*
   result for various specified tickets:
   EURUSD Order 0.50/0.50 Buy Limit #2812894647 Canceled by Client at 2024.09.04 19:0
   EURUSD Order 0.10/0 Sell #1753011743 Filled by Take Profit at 2023.06.12 17:04:20.
   GBPUSD Order 0.10/0 Buy #1819621374 Filled by Client at 2023.07.24 06:16:25.746
   */
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
//+------------------------------------------------------------------+
//| Return the order status description                              |
//+------------------------------------------------------------------+
string OrderStateDescription(ENUM_ORDER_STATE state)
  {
   switch(state)
     {
      case ORDER_STATE_STARTED         :  return("Started");
      case ORDER_STATE_PLACED          :  return("Placed");
      case ORDER_STATE_CANCELED        :  return("Canceled");
      case ORDER_STATE_PARTIAL         :  return("Partial");
      case ORDER_STATE_FILLED          :  return("Filled");
      case ORDER_STATE_REJECTED        :  return("Rejected");
      case ORDER_STATE_EXPIRED         :  return("Expired");
      case ORDER_STATE_REQUEST_ADD     :  return("Request Add");
      case ORDER_STATE_REQUEST_MODIFY  :  return("Request Modify");
      case ORDER_STATE_REQUEST_CANCEL  :  return("Request Cancel");
      default                          :  return("Unknown state: "+(string)state);
     }
  }
//+------------------------------------------------------------------+
//| Return the order placement reason description                    |
//+------------------------------------------------------------------+
string OrderReasonDescription(const ENUM_ORDER_REASON reason)
  {
   switch(reason)
     {
      case ORDER_REASON_CLIENT   :  return("Client");
      case ORDER_REASON_MOBILE   :  return("Mobile");
      case ORDER_REASON_WEB      :  return("Web");
      case ORDER_REASON_EXPERT   :  return("Expert");
      case ORDER_REASON_SL       :  return("Stop Loss");
      case ORDER_REASON_TP       :  return("Take Profit");
      case ORDER_REASON_SO       :  return("Stop Out");
      default                    :  return("Unknown reason: "+(string)reason);
     }
  }
//+------------------------------------------------------------------+
//| Return time with milliseconds                                    |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
```

See also

HistorySelect(), HistoryOrderGetTicket(), Order Properties

HistoryOrdersTotal

Returns the number of orders in the history. Prior to calling HistoryOrdersTotal(), first it is necessary to receive the history of deals and orders using the HistorySelect() or HistorySelectByPosition() function.

```
int  HistoryOrdersTotal();
```

Return Value

Value of the int type.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request all the existing history on the account
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }
     
//--- get the number of orders in the list and display it in the journal
   int total=HistoryOrdersTotal();
   Print("Number of historical orders on the account: ", total);
   /*
   result:
   Number of historical orders on the account: 496
   */
  }
```

See also

HistorySelect(), HistoryOrderSelect(), HistoryOrderGetTicket(), Order Properties

HistoryOrderGetTicket

Return the ticket of a corresponding order in the history. Prior to calling HistoryOrderGetTicket(), first it is necessary to receive the history of deals and orders using the HistorySelect() or HistorySelectByPosition() function.

```
ulong  HistoryOrderGetTicket(
   int  index      // Number in the list of orders
   );
```

Parameters

index

[in] Number of the order in the list of orders.

Return Value

Value of the ulong type. If the function fails, 0 is returned.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
void OnStart()
  {
   datetime from=0;
   datetime to=TimeCurrent();
//--- request the entire history
   HistorySelect(from,to);
//--- variables for returning values from order properties
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   datetime time_done;
   string   symbol;
   string   type;
   long     order_magic;
   long     positionID;
//--- number of current pending orders
   uint     total=HistoryOrdersTotal();
//--- go through orders in a loop
   for(uint i=0;i<total;i++)
     {
      //--- return order ticket by its position in the list
      if((ticket=HistoryOrderGetTicket(i))>0)
        {
         //--- return order properties
         open_price    =HistoryOrderGetDouble(ticket,ORDER_PRICE_OPEN);
         time_setup    =(datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_SETUP);
         time_done     =(datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_DONE);
         symbol        =HistoryOrderGetString(ticket,ORDER_SYMBOL);
         order_magic   =HistoryOrderGetInteger(ticket,ORDER_MAGIC);
         positionID    =HistoryOrderGetInteger(ticket,ORDER_POSITION_ID);
         initial_volume=HistoryOrderGetDouble(ticket,ORDER_VOLUME_INITIAL);
         type          =GetOrderType(HistoryOrderGetInteger(ticket,ORDER_TYPE));
         //--- prepare and show information about the order
         printf("#ticket %d %s %G %s at %G was set up at %s => done at %s, pos ID=%d",
                ticket,                  // order ticket
                type,                    // type
                initial_volume,          // placed volume
                symbol,                  // symbol
                open_price,              // specified open price
                TimeToString(time_setup),// time of order placing
                TimeToString(time_done), // time of order execution or deletion
                positionID               // ID of a position , to which the deal of th
                );
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| Returns the string name of the order type                        |
//+------------------------------------------------------------------+
string GetOrderType(long type)
  {
   string str_type="unknown operation";
   switch(type)
     {
      case (ORDER_TYPE_BUY):            return("buy");
      case (ORDER_TYPE_SELL):           return("sell");
      case (ORDER_TYPE_BUY_LIMIT):      return("buy limit");
      case (ORDER_TYPE_SELL_LIMIT):     return("sell limit");
      case (ORDER_TYPE_BUY_STOP):       return("buy stop");
      case (ORDER_TYPE_SELL_STOP):      return("sell stop");
      case (ORDER_TYPE_BUY_STOP_LIMIT): return("buy stop limit");
      case (ORDER_TYPE_SELL_STOP_LIMIT):return("sell stop limit");
     }
   return(str_type);
  }
```

See also

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Order Properties

HistoryOrderGetDouble

Returns the requested order property. The order property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  HistoryOrderGetDouble(
   ulong                       ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_DOUBLE  property_id        // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryOrderGetDouble(
   ulong                       ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_DOUBLE  property_id,       // Property identifier
   double&                     double_var         // Here we accept the property value
   );
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_DOUBLE enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the double type.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request deal and order history
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- in a loop by the list of all historical orders on the account
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the order ticket in the list by the loop index
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the order type and display the header for the list of real properties
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket,
      PrintFormat("Double properties of an history order %s #%I64u:", type, ticket);

      //--- print all the real properties of the selected order under the header
      HistoryOrderPropertiesDoublePrint(ticket, 16);
     }
   /*
   result:
   Double properties of an history order Sell #2810847541:
   Volume initial: 0.50
   Volume current: 0.00
   Price open:     1.10491
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10491
   StopLimit:      0.00000
   Double properties of an history order Buy Limit #2811003507:
   Volume initial: 1.00
   Volume current: 1.00
   Price open:     1.10547
   StopLoss:       0.00000
   TakeProfit:     0.00000
   Price current:  1.10591
   StopLimit:      0.00000
   */
  }
//+------------------------------------------------------------------+
//| Display real properties of the                                   |
//| selected historical order in the journal                         |
//+------------------------------------------------------------------+
void HistoryOrderPropertiesDoublePrint(const long ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;
```

```
//--- get the order symbol and the number of decimal places for the symbol
   string symbol = HistoryOrderGetString(ticket, ORDER_SYMBOL);
   int    digits = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Volume initial:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the initial volume when placing an order with a header of the sp
   if(!HistoryOrderGetDouble(ticket, ORDER_VOLUME_INITIAL, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the unfulfilled order volume in the journal
   header="Volume current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_VOLUME_CURRENT, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the price, specified in the order, in the journal
   header="Price open:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_OPEN, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the StopLoss level in the journal
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the TakeProfit level in the journal
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the current price by the order symbol in the journal
   header="Price current:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_CURRENT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);
```

```
//--- display the Limit order price, when StopLimit order is activated, in the journal
   header="StopLimit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetDouble(ticket, ORDER_PRICE_STOPLIMIT, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type");
     }
  }
```

See also

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Order Properties

HistoryOrderGetInteger

Returns the requested property of an order. The order property must be of datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  HistoryOrderGetInteger(
   ulong                          ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_INTEGER   property_id        // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryOrderGetInteger(
   ulong                          ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_INTEGER   property_id,       // Property identifier
   long&                          long_var           // Here we accept the property valu
   );
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_INTEGER enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the long type.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- receive the last order's ticket from week's trading history
   ulong last_order=GetLastOrderTicket();
   if(HistoryOrderSelect(last_order))
```

```
     {
      //--- time of placing an order in milliseconds since 01.01.1970
      long time_setup_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_SETUP_MSC);
      PrintFormat("Order #%d ORDER_TIME_SETUP_MSC=%i64 => %s",
                  last_order,time_setup_msc,TimeToString(time_setup_msc/1000));
      //--- order execution/cancellation time in milliseconds since 01.01.1970
      long  time_done_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_DONE_MSC);
      PrintFormat("Order #%d ORDER_TIME_DONE_MSC=%i64 => %s",
                  last_order,time_done_msc,TimeToString(time_done_msc/1000));
     }
   else // notify on failure
      PrintFormat("HistoryOrderSelect() failed for #%d. Eror code=%d",
                  last_order,GetLastError());

//---
  }
//+------------------------------------------------------------------+
//| Returns the last order ticket in history or -1                   |
//+------------------------------------------------------------------+
ulong GetLastOrderTicket()
  {
//--- request history for the last 7 days
   if(!GetTradeHistory(7))
     {
      //--- notify on unsuccessful call and return -1
      Print(__FUNCTION__," HistorySelect() returned false");
      return -1;
     }
//---
   ulong first_order,last_order,orders=HistoryOrdersTotal();
//--- work with orders if there are any
   if(orders>0)
     {
      Print("Orders = ",orders);
      first_order=HistoryOrderGetTicket(0);
      PrintFormat("first_order = %d",first_order);
      if(orders>1)
        {
         last_order=HistoryOrderGetTicket((int)orders-1);
         PrintFormat("last_order = %d",last_order);
         return last_order;
        }
      return first_order;
     }
//--- no order found, return -1
   return -1;
  }
//+--------------------------------------------------------------------------+
//| Requests history for the last days and returns false in case of failure  |
//+------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- set a week period to request trade history
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- make a request and check the result
   if(!HistorySelect(from,to))
     {
      Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
      return false;
     }
//--- history received successfully
   return true;
  }
```

See also

[HistorySelect()](), [HistoryOrdersTotal()](), [HistoryOrderSelect()](), [Order Properties]()

HistoryOrderGetString

Returns the requested property of an order. The order property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  HistoryOrderGetString(
   ulong                        ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_STRING   property_id        // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryOrderGetString(
   ulong                        ticket_number,     // Ticket
   ENUM_ORDER_PROPERTY_STRING   property_id,       // Property identifier
   string&                      string_var         // Here we accept the property value
   );
```

Parameters

ticket_number

[in] Order ticket.

property_id

[in] Identifier of the order property. The value can be one of the values of the ENUM_ORDER_PROPERTY_STRING enumeration.

string_var

[out] Variable of the string type.

Return Value

Value of the string type.

Note

Do not confuse orders of a trading history with current pending orders that appear on the "Trade" tab of the "Toolbox" bar. The list of orders that were canceled or have led to a transaction, can be viewed in the "History" tab of "Toolbox" of the client terminal.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request deal and order history
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- in a loop by the list of all historical orders on the account
   int total=HistoryOrdersTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the order ticket in the list by the loop index
      ulong ticket=HistoryOrderGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the order type and display the header for the list of string propertie
      string type=OrderTypeDescription((ENUM_ORDER_TYPE)HistoryOrderGetInteger(ticket,
      PrintFormat("String properties of an history order %s #%I64u:", type, ticket);

      //--- print all the string properties of the selected order under the header
      HistoryOrderPropertiesStringPrint(ticket, 16);
     }
   /*
   result:
   String properties of an history order Buy #2646074112:
   Comment:        [tp 1.09137]
   Symbol:         EURUSD
   External ID:
   String properties of an history order Buy #2646131906:
   Comment:
   Symbol:         EURUSD
   External ID:
   */
  }
//+------------------------------------------------------------------+
//| Display string properties of the                                 |
//| selected historical order in the journal                         |
//+------------------------------------------------------------------+
void HistoryOrderPropertiesStringPrint(const long ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value="";

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the comment with the specified header width in the journal
   if(!HistoryOrderGetString(ticket, ORDER_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- display a symbol the order has been placed for in the journal
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetString(ticket, ORDER_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- display the order ID in an external system in the journal
   header="External ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryOrderGetString(ticket, ORDER_EXTERNAL_ID, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
  }
//+------------------------------------------------------------------+
//| Return the order type description                                |
//+------------------------------------------------------------------+
string OrderTypeDescription(const ENUM_ORDER_TYPE type)
  {
   switch(type)
     {
      case ORDER_TYPE_BUY              :  return("Buy");
      case ORDER_TYPE_SELL             :  return("Sell");
      case ORDER_TYPE_BUY_LIMIT        :  return("Buy Limit");
      case ORDER_TYPE_SELL_LIMIT       :  return("Sell Limit");
      case ORDER_TYPE_BUY_STOP         :  return("Buy Stop");
      case ORDER_TYPE_SELL_STOP        :  return("Sell Stop");
      case ORDER_TYPE_BUY_STOP_LIMIT   :  return("Buy Stop Limit");
      case ORDER_TYPE_SELL_STOP_LIMIT  :  return("Sell Stop Limit");
      default                          :  return("Unknown order type: "+(string)type);
     }
  }
```

See also

[HistorySelect()](), [HistoryOrdersTotal()](), [HistoryOrderSelect()](), [Order Properties]()

HistoryDealSelect

Selects a deal in the history for further calling it through appropriate functions. It returns true if the function has been successfully completed. Returns false if the function has failed. For more details on error call GetLastError().

```
bool  HistoryDealSelect(
   ulong  ticket      // Deal ticket
   );
```

Parameters

ticket

[in] Deal ticket.

Return Value

Returns true if successful, otherwise false.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

HistoryDealSelect() clears in a mql5-program the list of deals available for reference, and copies the single deal, if the execution of HistoryDealSelect() has been completed successfully. If you need to go through all deals selected by the HistorySelect() function, you should better use HistoryDealGetTicket().

Example:

```
#define   TICKET    2620919264   // ticket of any known deal, for example, from the te

long      ExtTicket=TICKET;      // set the specified ticket to the variable from the
                                 // or handle deals in the OnTradeTransaction() handle
//+------------------------------------------------------------------+
//| Expert TradeTransaction handler                                  |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction& trans,
                        const MqlTradeRequest& request,
                        const MqlTradeResult& result)
  {
   //--- if a transaction is adding a deal to history
   if(trans.type==TRADE_TRANSACTION_DEAL_ADD)
     {
      //--- select a deal by ticket, get its data and display the deal description in
      HistoryDealSelectProcess(trans.deal);
     }
  }

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- select a deal by ticket, get its data and display the deal description in the jo
   HistoryDealSelectProcess(ExtTicket);
   /*
   result:
   (Position ID #2645974677) EURUSD Deal Out 0.10 Buy #2620919264 by order #264602896
   */
  }
//+------------------------------------------------------------------+
//| Select a deal by ticket and print the deal data in the journal   |
//+------------------------------------------------------------------+
void HistoryDealSelectProcess(const ulong deal_ticket)
  {
//--- select a historical deal by the ticket specified in deal_ticket
   ResetLastError();
   if(!HistoryDealSelect(deal_ticket))
     {
      PrintFormat("HistoryDealSelect(%I64u) failed. Error %d", deal_ticket, GetLastErr
      return;
     }

//--- if a deal is successfully selected, get its data and display the deal descriptio
   ENUM_DEAL_TYPE     deal_type  = (ENUM_DEAL_TYPE)HistoryDealGetInteger(ExtTicket, DEA
   ENUM_DEAL_ENTRY    deal_entry = (ENUM_DEAL_ENTRY)HistoryDealGetInteger(ExtTicket, DE
   ENUM_DEAL_REASON   deal_reason= (ENUM_DEAL_REASON)HistoryDealGetInteger(ExtTicket, D
   long               deal_time  = HistoryDealGetInteger(ExtTicket, DEAL_TIME_MSC);
   long               deal_order = HistoryDealGetInteger(ExtTicket, DEAL_ORDER);
   long               deal_pos_id= HistoryDealGetInteger(ExtTicket, DEAL_POSITION_ID);
   string             deal_symbol= HistoryDealGetString(ExtTicket, DEAL_SYMBOL);
   double             deal_volume= HistoryDealGetDouble(ExtTicket, DEAL_VOLUME);
   double             deal_price = HistoryDealGetDouble(ExtTicket, DEAL_PRICE);
   int                digits     = (int)SymbolInfoInteger(deal_symbol, SYMBOL_DIGITS);

   PrintFormat("(Position ID #%I64d) %s Deal %s %.2f %s #%I64u by order #%I64d at %.*f
               deal_pos_id, deal_symbol, DealEntryDescription(deal_entry), deal_volume
               DealTypeDescription(deal_type), ExtTicket, deal_order, digits, deal_pri
               TimeMscToString(deal_time));
  }
//+------------------------------------------------------------------+
//| Return the deal type description                                 |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                  :  return("Buy");
      case DEAL_TYPE_SELL                 :  return("Sell");
      case DEAL_TYPE_BALANCE                 :  return("Balance");
      case DEAL_TYPE_CREDIT                  :  return("Credit");
      case DEAL_TYPE_CHARGE                  :  return("Additional charge");
      case DEAL_TYPE_CORRECTION              :  return("Correction");
      case DEAL_TYPE_BONUS                   :  return("Bonus");
      case DEAL_TYPE_COMMISSION              :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY        :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY      :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY  :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY:  return("Monthly agent commission");
      case DEAL_TYPE_INTEREST                :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED            :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED           :  return("Canceled sell deal");
      case DEAL_DIVIDEND                     :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED             :  return("Franked (non-taxable) dividen
      case DEAL_TAX                          :  return("Tax charges");
      default                                :  return("Unknown deal type: "+(string)
     }
  }
//+------------------------------------------------------------------+
//| Return position change method                                    |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN      :  return("In");
      case DEAL_ENTRY_OUT     :  return("Out");
      case DEAL_ENTRY_INOUT   :  return("Reverce");
      case DEAL_ENTRY_OUT_BY  :  return("Out by");
      case DEAL_ENTRY_STATE   :  return("Status record");
      default                 :  return("Unknown deal entry: "+(string)entry);
     }
  }
//+------------------------------------------------------------------+
//| Return time with milliseconds                                    |
//+------------------------------------------------------------------+
string TimeMscToString(const long time_msc, int flags=TIME_DATE|TIME_MINUTES|TIME_SEC
  {
   return(TimeToString(time_msc/1000, flags) + "." + IntegerToString(time_msc %1000, 
  }
```

See also

HistorySelect(), HistoryDealGetTicket(), Deal Properties

HistoryDealsTotal

Returns the number of deal in history. Prior to calling HistoryDealsTotal(), first it is necessary to receive the history of deals and orders using the HistorySelect() or HistorySelectByPosition() function.

```
int  HistoryDealsTotal();
```

Return Value

Value of the int type.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request all the existing history on the account
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- get the number of deals in the list and display it in the journal
   int total=HistoryDealsTotal();
   Print("Number of historical deals on the account: ", total);
   /*
   result:
   Number of historical deals on the account: 339
   */
  }
```

See also

HistorySelect(), HistoryDealGetTicket(), Deal Properties

HistoryDealGetTicket

The function selects a deal for further processing and returns the deal ticket in history. Prior to calling HistoryDealGetTicket(), first it is necessary to receive the history of deals and orders using the HistorySelect() or HistorySelectByPosition() function.

```
ulong  HistoryDealGetTicket(
   int  index      // ticket deal
   );
```

Parameters

index

[in] Number of a deal in the list of deals

Return Value

Value of the ulong type. If the function fails, 0 is returned.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
void OnStart()
  {
   ulong deal_ticket;            // deal ticket
   ulong order_ticket;           // ticket of the order the deal was executed on
   datetime transaction_time;    // time of a deal execution
   long deal_type ;              // type of a trade operation
   long position_ID;             // position ID
   string deal_description;      // operation description
   double volume;                // operation volume
   string symbol;                // symbol of the deal
//--- set the start and end date to request the history of deals
   datetime from_date=0;         // from the very beginning
   datetime to_date=TimeCurrent();// till the current moment
//--- request the history of deals in the specified period
   HistorySelect(from_date,to_date);
//--- total number in the list of deals
   int deals=HistoryDealsTotal();
//--- now process each trade
   for(int i=0;i<deals;i++)
     {
      deal_ticket=               HistoryDealGetTicket(i);
      volume=                    HistoryDealGetDouble(deal_ticket,DEAL_VOLUME);
      transaction_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
      order_ticket=              HistoryDealGetInteger(deal_ticket,DEAL_ORDER);
      deal_type=                 HistoryDealGetInteger(deal_ticket,DEAL_TYPE);
      symbol=                    HistoryDealGetString(deal_ticket,DEAL_SYMBOL);
      position_ID=               HistoryDealGetInteger(deal_ticket,DEAL_POSITION_ID);
      deal_description=          GetDealDescription(deal_type,volume,symbol,order_tic
      //--- perform fine formatting for the deal number
      string print_index=StringFormat("% 3d",i);
      //--- show information on the deal
      Print(print_index+": deal #",deal_ticket," at ",transaction_time,deal_descripti
     }
  }
//+------------------------------------------------------------------+
//| Returns the string description of the operation                  |
//+------------------------------------------------------------------+
string GetDealDescription(long deal_type,double volume,string symbol,long ticket,long
  {
   string descr;
//---
   switch(deal_type)
     {
      case DEAL_TYPE_BALANCE:                  return ("balance");
      case DEAL_TYPE_CREDIT:                   return ("credit");
      case DEAL_TYPE_CHARGE:                   return ("charge");
      case DEAL_TYPE_CORRECTION:               return ("correction");
      case DEAL_TYPE_BUY:                      descr="buy"; break;
      case DEAL_TYPE_SELL:                     descr="sell"; break;
      case DEAL_TYPE_BONUS:                    return ("bonus");
      case DEAL_TYPE_COMMISSION:               return ("additional commission");
      case DEAL_TYPE_COMMISSION_DAILY:         return ("daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY:       return ("monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY:   return ("daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return ("monthly agent commission");
      case DEAL_TYPE_INTEREST:                 return ("interest rate");
      case DEAL_TYPE_BUY_CANCELED:             descr="cancelled buy deal"; break;
      case DEAL_TYPE_SELL_CANCELED:            descr="cancelled sell deal"; break;
     }
   descr=StringFormat("%s %G %s (order #%d, position ID %d)",
                      descr,  // current description
                      volume, // deal volume
                      symbol, // deal symbol
                      ticket, // ticket of the order that caused the deal
                      pos_ID  // ID of a position, in which the deal is included
                      );
   return(descr);
//---
  }
```

See also

[HistorySelect()](), [HistoryDealsTotal()](), [HistoryDealSelect()](), [Deal Properties]()

HistoryDealGetDouble

Returns the requested property of a deal. The deal property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  HistoryDealGetDouble(
   ulong                       ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_DOUBLE   property_id         // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryDealGetDouble(
   ulong                       ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_DOUBLE   property_id,        // Property identifier
   double&                     double_var          // Here we accept the property value
   );
```

Parameters

ticket_number

[in] Deal ticket.

property_id

[in] Identifier of a deal property. The value can be one of the values of the ENUM_DEAL_PROPERTY_DOUBLE enumeration.

double_var

[out] Variable of the double type that accepts the value of the requested property.

Return Value

Value of the double type.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request deal and order history
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- in a loop by the list of deals in the account history
   int total=HistoryDealsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the ticket of the next deal (the deal is automatically selected to get
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the type and direction of the deal and display the header for the list
      string type=DealTypeDescription((ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, DE
      string entry=DealEntryDescription((ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket,
      PrintFormat("Double properties of an deal %s entry %s #%I64u:", type, entry, tic

      //--- print all the real properties of the selected deal under the header
      HistoryDealPropertiesDoublePrint(ticket, 12);
     }
   /*
   real:
   Double properties of an deal Buy entry In #2785070622:
   Volume:     0.50
   Price:      1.10480
   Commission: 0.00
   Swap:       0.00
   Profit:     0.00 USD
   Fee:        0.00
   StopLoss:   0.00000
   TakeProfit: 0.00000
   Double properties of an deal Sell entry Out #2785071538:
   Volume:     0.50
   Price:      1.10491
   Commission: 0.00
   Swap:       0.00
   Profit:     5.50 USD
   Fee:        0.00
   StopLoss:   0.00000
   TakeProfit: 0.00000
   */
  }
//+------------------------------------------------------------------+
//| Display real properties of the selected deal in the journal      |
//+------------------------------------------------------------------+
void HistoryDealPropertiesDoublePrint(const ulong ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   double value=0;

//--- get the deal symbol, profit currency and the number of decimal places for the sy
   string symbol   = HistoryDealGetString(ticket, DEAL_SYMBOL);
   string currency= SymbolInfoString(symbol, SYMBOL_CURRENCY_PROFIT);
   int    digits   = (int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Volume:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the deal volume with the specified header width in the journal
   if(!HistoryDealGetDouble(ticket, DEAL_VOLUME, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the deal price in the journal
   header="Price:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_PRICE, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the deal commission in the journal
   header="Commission:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_COMMISSION, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the accumulated swap in the journal when closing
   header="Swap:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_SWAP, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the deal financial result in the journal
   header="Profit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_PROFIT, value))
      return;
   PrintFormat("%-*s%-.2f %s", w, header, value, currency);

//--- display the deal fee in the journal
   header="Fee:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_FEE, value))
      return;
   PrintFormat("%-*s%-.2f", w, header, value);

//--- display the StopLoss level in the journal
   header="StopLoss:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_SL, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);

//--- display the TakeProfit level in the journal
   header="TakeProfit:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetDouble(ticket, DEAL_TP, value))
      return;
   PrintFormat("%-*s%-.*f", w, header, digits, value);
  }
//+------------------------------------------------------------------+
//| Return the deal type description                                 |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                     :  return("Buy");
      case DEAL_TYPE_SELL                    :  return("Sell");
      case DEAL_TYPE_BALANCE                 :  return("Balance");
      case DEAL_TYPE_CREDIT                  :  return("Credit");
      case DEAL_TYPE_CHARGE                  :  return("Additional charge");
      case DEAL_TYPE_CORRECTION              :  return("Correction");
      case DEAL_TYPE_BONUS                   :  return("Bonus");
      case DEAL_TYPE_COMMISSION              :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY        :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY      :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY  :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY:  return("Monthly agent commission");
      case DEAL_TYPE_INTEREST                :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED            :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED           :  return("Canceled sell deal");
      case DEAL_DIVIDEND                     :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED             :  return("Franked (non-taxable) dividen
      case DEAL_TAX                          :  return("Tax charges");
      default                                :  return("Unknown deal type: "+(string)
     }
  }
//+------------------------------------------------------------------+
//| Return position change method                                    |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
     {
      case DEAL_ENTRY_IN       :  return("In");
      case DEAL_ENTRY_OUT      :  return("Out");
      case DEAL_ENTRY_INOUT    :  return("Reverce");
      case DEAL_ENTRY_OUT_BY   :  return("Out by");
      case DEAL_ENTRY_STATE    :  return("Status record");
      default                  :  return("Unknown deal entry: "+(string)entry);
     }
  }
```

See also

HistorySelect(), HistoryDealsTotal(), HistoryDealGetTicket(), Deal Properties

HistoryDealGetInteger

Returns the requested property of a deal. The deal property must be of the datetime, int type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  HistoryDealGetInteger(
   ulong                        ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_INTEGER  property_id         // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryDealGetInteger(
   ulong                        ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_INTEGER  property_id,        // Property identifier
   long&                        long_var            // Here we accept the property valu
   );
```

Parameters

ticket_number

[in] Trade ticket.

property_id

[in] Identifier of the deal property. The value can be one of the values of the ENUM_DEAL_PROPERTY_INTEGER enumeration.

long_var

[out] Variable of the long type that accepts the value of the requested property.

Return Value

Value of the long type.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- receive the last deal's ticket from week's trading history
   ulong last_deal=GetLastDealTicket();
   if(HistoryDealSelect(last_deal))
     {
      //--- time of deal execution in milliseconds since 01.01.1970
      long deal_time_msc=HistoryDealGetInteger(last_deal,DEAL_TIME_MSC);
      PrintFormat("Deal #%d DEAL_TIME_MSC=%i64 => %s",
                  last_deal,deal_time_msc,TimeToString(deal_time_msc/1000));
     }
   else
      PrintFormat("HistoryDealSelect() failed for #%d. Eror code=%d",
                  last_deal,GetLastError());
//---
  }
//+------------------------------------------------------------------+
//| Returns the last deal ticket in history or -1                    |
//+------------------------------------------------------------------+
ulong GetLastDealTicket()
  {
//--- request history for the last 7 days
   if(!GetTradeHistory(7))
     {
      //--- notify on unsuccessful call and return -1
      Print(__FUNCTION__," HistorySelect() returned false");
      return -1;
     }
//---
   ulong first_deal,last_deal,deals=HistoryDealsTotal();
//--- work with orders if there are any
   if(deals>0)
     {
      Print("Deals = ",deals);
      first_deal=HistoryDealGetTicket(0);
      PrintFormat("first_deal = %d",first_deal);
      if(deals>1)
        {
         last_deal=HistoryDealGetTicket((int)deals-1);
         PrintFormat("last_deal = %d",last_deal);
         return last_deal;
        }
      return first_deal;
     }
//--- no deal found, return -1
   return -1;
  }
//+--------------------------------------------------------------------------+
//| Requests history for the last days and returns false in case of failure   |
//+--------------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- set a week period to request trade history
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- make a request and check the result
   if(!HistorySelect(from,to))
     {
      Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
      return false;
     }
//--- history received successfully
   return true;
  }
```

See also

HistoryDealsTotal(), HistorySelect(), HistoryDealGetTicket(), Deal Properties

HistoryDealGetString

Returns the requested property of a deal. The deal property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  HistoryDealGetString(
   ulong                        ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_STRING    property_id         // Property identifier
   );
```

2. Returns true or false, depending on the success of the function. If successful, the value of the property is placed into a target variable passed by reference by the last parameter.

```
bool  HistoryDealGetString(
   ulong                        ticket_number,      // Ticket
   ENUM_DEAL_PROPERTY_STRING    property_id,        // Property identifier
   string&                      string_var          // Here we accept the property value
   );
```

Parameters

ticket_number

[in] Deal ticket.

property_id

[in] Identifier of the deal property. The value can be one of the values of the ENUM_DEAL_PROPERTY_STRING enumeration.

string_var

[out] Variable of the string type that accepts the value of the requested property.

Return Value

Value of the string type.

Note

Do not confuse orders, deals and positions. Each deal is the result of the execution of an order, each position is the summary result of one or more deals.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- request deal and order history
   if(!HistorySelect(0, TimeCurrent()))
     {
      Print("HistorySelect() failed. Error ", GetLastError());
      return;
     }

//--- in a loop by the list of deals in the account history
   int total=HistoryDealsTotal();
   for(int i=0; i<total; i++)
     {
      //--- get the ticket of the next deal (the deal is automatically selected to get
      ulong ticket=HistoryDealGetTicket(i);
      if(ticket==0)
         continue;

      //--- get the type and direction of the deal and display the header for the list
      string type=DealTypeDescription((ENUM_DEAL_TYPE)HistoryDealGetInteger(ticket, D
      string entry=DealEntryDescription((ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket,
      PrintFormat("String properties of an deal %s entry %s #%I64u:", type, entry, tic

      //--- print all the real properties of the selected deal under the header
      HistoryDealPropertiesStringPrint(ticket, 13);
     }
   /*
   result:
   String properties of an deal Buy entry In #2785021084:
   Symbol:      EURUSD
   Comment:     Test PositionGetString
   Extarnal ID:
   String properties of an deal Buy entry Out #2497993663:
   Symbol:      EURUSD
   Comment:     [tp 1.08639]
   Extarnal ID:
   */
  }
//+------------------------------------------------------------------+
//| Display string properties of the selected deal in the journal    |
//+------------------------------------------------------------------+
void HistoryDealPropertiesStringPrint(const ulong ticket, const uint header_width=0)
  {
   uint   w=0;
   string header="";
   string value ="";

//--- define the header text and the width of the header field
//--- if the header width is passed to the function equal to zero, then the width wil
   header="Symbol:";
   w=(header_width==0 ? header.Length()+1 : header_width);
//--- get and display the deal symbol with the specified header width in the journal
   if(!HistoryDealGetString(ticket, DEAL_SYMBOL, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
```

```
//--- display the deal comment in the journal
   header="Comment:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetString(ticket, DEAL_COMMENT, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);

//--- display the deal ID in an external trading system
   header="Extarnal ID:";
   w=(header_width==0 ? header.Length()+1 : header_width);
   if(!HistoryDealGetString(ticket, DEAL_EXTERNAL_ID, value))
      return;
   PrintFormat("%-*s%-s", w, header, value);
  }
//+------------------------------------------------------------------+
//| Return the deal type description                                  |
//+------------------------------------------------------------------+
string DealTypeDescription(const ENUM_DEAL_TYPE type)
  {
   switch(type)
     {
      case DEAL_TYPE_BUY                     :  return("Buy");
      case DEAL_TYPE_SELL                    :  return("Sell");
      case DEAL_TYPE_BALANCE                 :  return("Balance");
      case DEAL_TYPE_CREDIT                  :  return("Credit");
      case DEAL_TYPE_CHARGE                  :  return("Additional charge");
      case DEAL_TYPE_CORRECTION              :  return("Correction");
      case DEAL_TYPE_BONUS                   :  return("Bonus");
      case DEAL_TYPE_COMMISSION              :  return("Additional commission");
      case DEAL_TYPE_COMMISSION_DAILY        :  return("Daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY      :  return("Monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY  :  return("Daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY:  return("Monthly agent commission");
      case DEAL_TYPE_INTEREST                :  return("Interest rate");
      case DEAL_TYPE_BUY_CANCELED            :  return("Canceled buy deal");
      case DEAL_TYPE_SELL_CANCELED           :  return("Canceled sell deal");
      case DEAL_DIVIDEND                     :  return("Dividend operations");
      case DEAL_DIVIDEND_FRANKED             :  return("Franked (non-taxable) dividen
      case DEAL_TAX                          :  return("Tax charges");
      default                                :  return("Unknown deal type: "+(string)t
     }
  }
//+------------------------------------------------------------------+
//| Return position change method                                     |
//+------------------------------------------------------------------+
string DealEntryDescription(const ENUM_DEAL_ENTRY entry)
  {
   switch(entry)
```

```
      {
       case DEAL_ENTRY_IN       :  return("In");
       case DEAL_ENTRY_OUT      :  return("Out");
       case DEAL_ENTRY_INOUT    :  return("Reverce");
       case DEAL_ENTRY_OUT_BY   :  return("Out by");
       case DEAL_ENTRY_STATE    :  return("Status record");
       default                  :  return("Unknown deal entry: "+(string)entry);
      }
   }
```

See also

HistoryDealsTotal(), HistorySelect(), HistoryDealGetTicket(), Deal Properties

Trade Signals

This is the group of functions intended for managing trade signals. The functions allow:

- get information about trade signals, available for copying,
- get and set the signal copy settings,
- subscribe and unsubscribe to the signal copying using MQL5 language functions.

Function	Action
SignalBaseGetDouble	Returns the value of double type property for selected signal
SignalBaseGetInteger	Returns the value of integer type property for selected signal
SignalBaseGetString	Returns the value of string type property for selected signal
SignalBaseSelect	Selects a signal from signals, available in terminal for further working with it
SignalBaseTotal	Returns the total amount of signals, available in terminal
SignalInfoGetDouble	Returns the value of double type property of signal copy settings
SignalInfoGetInteger	Returns the value of integer type property of signal copy settings
SignalInfoGetString	Returns the value of string type property of signal copy settings
SignalInfoSetDouble	Sets the value of double type property of signal copy settings
SignalInfoSetInteger	Sets the value of integer type property of signal copy settings
SignalSubscribe	Subscribes to the trading signal
SignalUnsubscribe	Cancels subscription

SignalBaseGetDouble

Returns the value of double type property for selected signal.

```
double  SignalBaseGetDouble(
   ENUM_SIGNAL_BASE_DOUBLE     property_id,     // property identifier
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_DOUBLE enumeration.

Return Value

The value of double type property of the selected signal.

SignalBaseGetInteger

Returns the value of integer type property for selected signal.

```
long  SignalBaseGetInteger(
   ENUM_SIGNAL_BASE_INTEGER     property_id,      // property identifier
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_INTEGER enumeration.

Return Value

The value of integer type property of the selected signal.

SignalBaseGetString

Returns the value of string type property for selected signal.

```
string  SignalBaseGetString(
   ENUM_SIGNAL_BASE_STRING     property_id,      // property identifier
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_STRING enumeration.

Return Value

The value of string type property of the selected signal.

SignalBaseSelect

Selects a signal from signals, available in terminal for further working with it.

```
bool  SignalBaseSelect(
   int      index      // signal index
   );
```

Parameters

index

[in] Signal index in base of trading signals.

Return Value

Returns true if successful, otherwise returns false. To read more about the error call GetLastError().

Example:

```
void OnStart()
  {
//--- get total amount of signals in the terminal
   int total=SignalBaseTotal();
//--- process all signals
   for(int i=0;i<total;i++)
     {
      //--- select the signal by index
      if(SignalBaseSelect(i))
        {
         //--- get signal properties
         long   id    =SignalBaseGetInteger(SIGNAL_BASE_ID);          // signal id
         long   pips  =SignalBaseGetInteger(SIGNAL_BASE_PIPS);        // profit in pip
         long   subscr=SignalBaseGetInteger(SIGNAL_BASE_SUBSCRIBERS); // number of sub
         string name  =SignalBaseGetString(SIGNAL_BASE_NAME);         // signal name
         double price =SignalBaseGetDouble(SIGNAL_BASE_PRICE);        // signal price
         string curr  =SignalBaseGetString(SIGNAL_BASE_CURRENCY);     // signal curren
         //--- print all profitable free signals with subscribers
         if(price==0.0 && pips>0 && subscr>0)
            PrintFormat("id=%d, name=\"%s\", currency=%s, pips=%d, subscribers=%d",id,
        }
      else PrintFormat("Error in call of SignalBaseSelect. Error code=%d",GetLastErro
     }
  }
```

SignalBaseTotal

Returns the total amount of signals, available in terminal.

```
int  SignalBaseTotal();
```

Return Value

The total amount of signals, available in terminal.

SignalInfoGetDouble

Returns the value of double type property of signal copy settings.

```
double  SignalInfoGetDouble(
   ENUM_SIGNAL_INFO_DOUBLE     property_id,      // property identifier
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_DOUBLE enumeration.

Return Value

The value of double type property of signal copy settings.

SignalInfoGetInteger

Returns the value of integer type property of signal copy settings.

```
long  SignalInfoGetInteger(
   ENUM_SIGNAL_INFO_INTEGER     property_id,      // property identifier
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_INTEGER enumeration.

Return Value

The value of integer type property of signal copy settings.

SignalInfoGetString

Returns the value of string type property of signal copy settings.

```
string  SignalInfoGetString(
   ENUM_SIGNAL_INFO_STRING     property_id,      // property identifier
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_STRING enumeration.

Return Value

The value of string type property of signal copy settings.

SignalInfoSetDouble

Sets the value of double type property of signal copy settings.

```
bool  SignalInfoSetDouble(
   ENUM_SIGNAL_INFO_DOUBLE      property_id,      // property identifier
   double                       value             // new value
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_DOUBLE enumeration.

value

[in] The value of signal copy settings property.

Return Value

Returns true if property has been changed, otherwise returns false. To read more about the error call GetLastError().

SignalInfoSetInteger

Sets the value of integer type property of signal copy settings.

```
bool  SignalInfoSetInteger(
   ENUM_SIGNAL_INFO_INTEGER      property_id,      // property identifier
   long                          value             // new value
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_INTEGER enumeration.

value

[in] The value of signal copy settings property.

Return Value

Returns true if property has been changed, otherwise returns false. To read more about the error call GetLastError().

SignalSubscribe

Subscribes to the trading signal.

```
bool  SignalSubscribe(
   long      signal_id      // signal id
   );
```

Parameters

signal_id

[in] Signal identifier.

Return Value

Returns true if subscription was successful, otherwise returns false. To read more about the error call GetLastError().

SignalUnsubscribe

Cancels subscription.

```
bool  SignalUnsubscribe();
```

Return Value

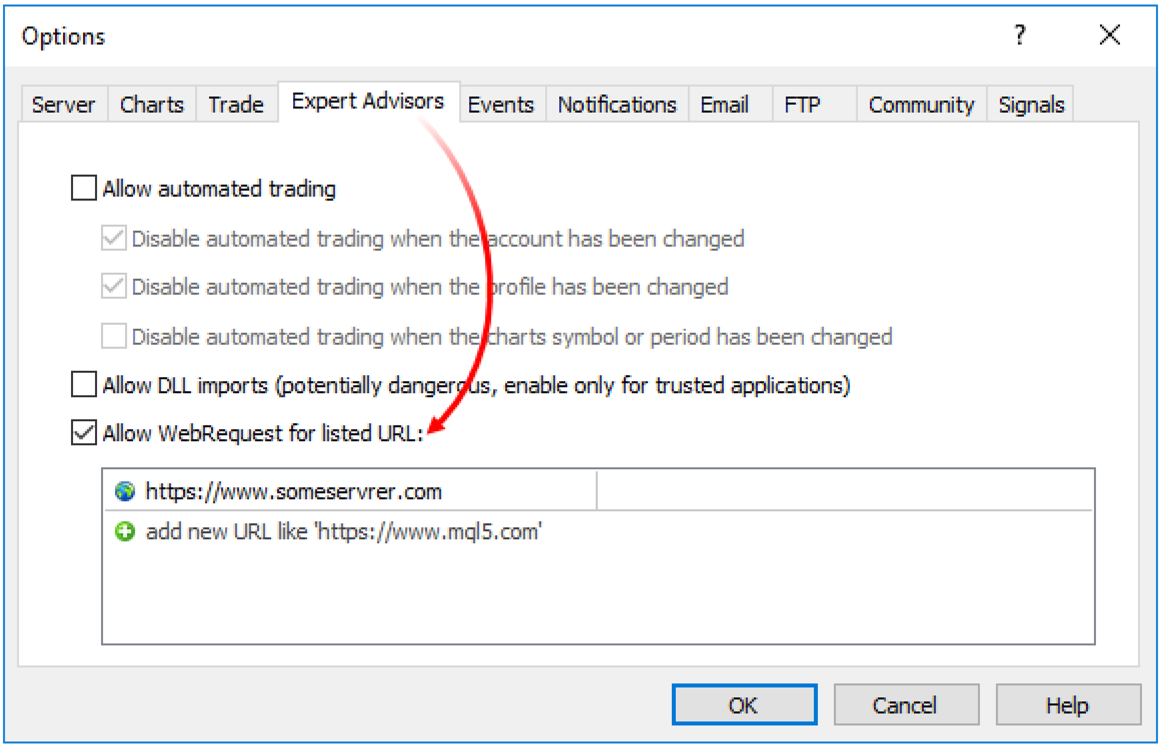
Returns true if subscription has been canceled successfully, otherwise returns false. To read more about the error call GetLastError().

Network functions

MQL5 programs can exchange data with remote servers, as well as send push notifications, emails and data via FTP.

- The Socket* group of functions allows establishing a TCP connection (including a secure TLS) with a remote host via system sockets. The operation principle is simple: create a socket, connect to the server and start reading and writing data.
- The WebRequest function is designed to work with web resources and allows sending HTTP requests (including GET and POST) easily.
- SendFTP, SendMail and SendNotification are more simple functions for sending files, emails and mobile notifications.

For end-user security, the list of allowed IP addresses is implemented on the client terminal side. The list contains IP addresses the MQL5 program is allowed to connect to via the Socket* and WebRequest functions. For example, if the program needs to connect to https://www.someserver.com, this address should be explicitly indicated by a terminal user in the list. An address cannot be added programmatically.



☐ Allow automated trading

☑ Disable automated trading when the account has been changed

☑ Disable automated trading when the profile has been changed

☐ Disable automated trading when the charts symbol or period has been changed

☐ Allow DLL imports (potentially dangerous, enable only for trusted applications)

☑ Allow WebRequest for listed URL:

https://www.someservrer.com

add new URL like 'https://www.mql5.com'

Add an explicit message to the MQL5 program to notify a user of the need for additional configuration. You can do that via #property description, Alert or Print.

Function	Action
SocketCreate	Create a socket with specified flags and return its handle
SocketClose	Close a socket
SocketConnect	Connect to the server with timeout control
SocketIsConnected	Checks if the socket is currently connected

Function	Action
SocketIsReadable	Get a number of bytes that can be read from a socket
SocketIsWritable	Check whether data can be written to a socket at the current time
SocketTimeouts	Set timeouts for receiving and sending data for a socket system object
SocketRead	Read data from a socket
SocketSend	Write data to a socket
SocketTlsHandshake	Initiate secure TLS (SSL) connection to a specified host via TLS Handshake protocol
SocketTlsCertificate	Get data on the certificate used to secure network connection
SocketTlsRead	Read data from secure TLS connection
SocketTlsReadAvailable	Read all available data from secure TLS connection
SocketTlsSend	Send data via secure TLS connection
WebRequest	Send an HTTP request to a specified server
SendFTP	Send a file to an address specified on the FTP tab
SendMail	Send an email to an address specified in the Email tab of the options window
SendNotification	Send push notifications to mobile terminals whose MetaQuotes IDs are specified in the Notifications tab

SocketCreate

Create a socket with specified flags and return its handle.

```
int  SocketCreate(
   uint      flags          // flags
   );
```

Parameters

flags

[in] Combination of flags defining the mode of working with a socket. Currently, only one flag is supported – SOCKET_DEFAULT.

Return Value

In case of a successful socket creation, return its handle, otherwise INVALID_HANDLE.

Notes

To free up computer memory from an unused socket, call SocketClose for it.

You can create a maximum of 128 sockets from one MQL5 program. If the limit is exceeded, the error 5271 (ERR_NETSOCKET_TOO_MANY_OPENED) is written to _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
```

```
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- close a socket after using
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

SocketClose

Close a socket.

```
bool  SocketClose(
   const int  socket      // socket handle
   );
```

Parameters

socket

[in] Handle of a socket to be closed. The handle is returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

Return Value

Returns true if successful, otherwise false.

Note

If a connection via SocketConnect was previously created for a socket, it is discontinued.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
```

```
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- close a socket after using
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

SocketConnect

Connect to the server with timeout control.

```
bool  SocketConnect(
   int            socket,                // socket
   const string   server,                // connection address
   uint           port,                  // connection port
   uint           timeout_receive_ms     // connection timeout
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

server

[in] Domain name of the server you want to connect to or its IP address.

port

[in] Connection port number.

timeout_receive_ms

[in] Connection timeout in milliseconds. If connection is not established within that time interval, attempts are stopped.

Return Value

If connection is successful, return true, otherwise false.

Note

Connection address should be added to the list of allowed ones on the client terminal side (Tools \ Options \ Expert Advisors).

If connection fails, error 5272 (ERR_NETSOCKET_CANNOT_CONNECT) is written to _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs
```

```
input string Address="www.mql5.com";
input int    Port    =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
         }
       //--- close a socket after using
       SocketClose(socket);
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

SocketIsConnected

Checks if the socket is currently connected.

```
bool  SocketIsConnected(
   const int  socket      // socket handle
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate() function. When an incorrect handle is passed to _LastError, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is activated.

Return Value

Returns true if the socket is connected, otherwise - false.

Note

The SocketIsConnected() function allows checking the current socket connection status.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                            SocketIsConnected.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version   "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address     ="www.mql5.com";
input int    Port        =80;
input bool   CloseSocket=true;
bool         ExtTLS      =false;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- create a socket and get its handle
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
```

```
     {
      //--- if all is well, connect
      if(SocketConnect(socket,Address,Port,1000))
        {
         PrintFormat("Established connection to %s:%d",Address,Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is protected by certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:      ",subject);
            Print("   Issuer:     ",issuer);
            Print("   Number:     ",serial);
            Print("   Print:      ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
        }
      //--- if the flag is set, close the socket after use
      if(CloseSocket)
         SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());

//--- check server connection
   bool connected=SocketIsConnected(socket);
//--- terminate operation if no connection
   if(!connected)
     {
      Print("No connection to server");
     }
//--- if server connection is present
```

```
   else
     {
      Print("Connection to the server is available\nThe connection needs to be closed.
      //--- close the socket and check the connection status again
      SocketClose(socket);
      connected=SocketIsConnected(socket);
     }

//--- display the current server connection status
   Print("Currently connected: ",(connected ? "opened" : "closed"));
   /*
   result in case CloseSocket = true:
   No connection to server
   Currently connected: closed

   result in case CloseSocket = false:
   Connection to the server is available
   The connection needs to be closed. Closing
   Currently connected: closed
   */
  }
//+------------------------------------------------------------------+
//| Send command to server                                           |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
//--- convert the string to a character array, discard the terminating zero
   char req[];
   int  len=StringToCharArray(request,req)-1;

   if(len<0)
      return(false);
//--- if a secure TLS connection via port 443 is used
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if a regular TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;
//--- read data from the socket while there is some, but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);

      if(len)
        {
         int rsp_len;
         //--- different read commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout_ms);
         //--- parse response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- display response header only
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
            //--- update read timeout expiration time
            timeout_check=GetTickCount64()+timeout_ms;
           }
        }
     }
   while(GetTickCount64()<timeout_check && !IsStopped());

   return(false);
  }
```

See also

SocketConnect, SocketIsWritable, SocketCreate, SocketClose

SocketIsReadable

Get a number of bytes that can be read from a socket.

```
uint  SocketIsReadable(
   const int  socket      // socket handle
   );
```

Parameters

socket

[in] Socket handle returned by the [SocketCreate](#) function. When an incorrect handle is passed to [_LastError](#), the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is activated.

Return Value

Number of bytes that can be read. In case of an error, 0 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via [SocketConnect](#) is discontinued.

Before calling [SocketRead](#), check if the socket features data for reading. Otherwise, if there are no data, the [SocketRead](#) function waits for data within timeout_ms delaying the program execution.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, [GetLastError()](#) returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- close a socket after using
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

SocketIsWritable

Check whether data can be written to a socket at the current time.

```
bool  SocketIsWritable(
   const int  socket      // socket handle
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

Return Value

Return true if writing is possible, otherwise false.

Note

This function allows you to check whether it is possible to write data to a socket right now.

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                             SocketIsWritable.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address    ="www.mql5.com";
input int    Port       =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- create a socket and get its handle
   int socket=SocketCreate();
//--- check the handle
```

```
if(socket!=INVALID_HANDLE)
  {
   //--- if all is well, connect
   if(SocketConnect(socket,Address,Port,1000))
     {
      PrintFormat("Established connection to %s:%d",Address,Port);

      string   subject,issuer,serial,thumbprint;
      datetime expiration;
      //--- if connection is protected by certificate, display its data
      if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
        {
         Print("TLS certificate:");
         Print("   Owner:      ",subject);
         Print("   Issuer:     ",issuer);
         Print("   Number:     ",serial);
         Print("   Print:      ",thumbprint);
         Print("   Expiration: ",expiration);
         ExtTLS=true;
        }
      //--- send GET request to the server
      if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r
        {
         Print("GET request sent");
         //--- read response
         if(!HTTPRecv(socket,1000))
            Print("Failed to get a response, error ",GetLastError());
        }
      else
         Print("Failed to send GET request, error ",GetLastError());
     }
   else
     {
      PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
     }
   //--- display the ability to write data to the socket at the current moment in t
   if(SocketIsWritable(socket))
      Print("At the current moment in time, writing data to the socket is possible'
   else
      Print("It is not possible to write data to the socket at the current time");
   //--- close the socket after use
   if(SocketClose(socket))
      Print("Now the socket is closed");
  }
else
   Print("Failed to create a socket, error ",GetLastError());
/*
result:
At the current moment in time, writing data to the socket is possible
```

```
   Socket is closed now
   */
  }
//+------------------------------------------------------------------+
//| Send command to server                                           |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
//--- convert the string to a character array, discard the terminating zero
   char req[];
   int  len=StringToCharArray(request,req)-1;

   if(len<0)
      return(false);
//--- if a secure TLS connection via port 443 is used
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if a regular TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;
//--- read data from the socket while there is some, but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);

      if(len)
        {
         int rsp_len;
         //--- different read commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout_ms);
         //--- parse response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- display response header only
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
            {
             Print("HTTP answer header received:");
             Print(StringSubstr(result,0,header_end));
             return(true);
            }
          //--- update read timeout expiration time
          timeout_check=GetTickCount64()+timeout_ms;
         }
      }
   }
 while(GetTickCount64()<timeout_check && !IsStopped());

 return(false);
}
```

SocketTimeouts

Set timeouts for receiving and sending data for a socket system object.

```
bool  SocketTimeouts(
   int            socket,               // socket
   uint           timeout_send_ms,      // data sending timeout
   uint           timeout_receive_ms    // data obtaining timeout
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

timeout_send_ms

[in] Data sending timeout in milliseconds.

timeout_receive_ms

[in] Data obtaining timeout in milliseconds.

Return Value

Returns true if successful, otherwise false.

Note

Do not confuse system object timeouts with the ones set when reading data via SocketRead. SocketTimeout sets timeouts once for a socket object in the operating system. These timeouts are to be applied to all functions for reading and sending data via this socket. In SocketRead, the timeout is set for a certain data reading operation.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                           SocketTimeouts.mq5 |
//|                             Copyright 2024, MetaQuotes Ltd. |
//|                                      https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version    "1.00"

#define   TIMEOUT_SEND      5000
#define   TIMEOUT_RECEIVE   5000
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- create a socket and get its handle
   int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- set timeouts for receiving and sending data for a socket system object
   if(SocketTimeouts(socket,TIMEOUT_SEND,TIMEOUT_RECEIVE))
      PrintFormat("timeouts were successfully set",socket,TIMEOUT_SEND,TIMEOUT_RECEIV
   else
      PrintFormat("SocketTimeouts(%d, %I64d, %I64d) failed. Error %d",socket,TIMEOUT_
//--- close socket
   SocketClose(socket);
  }
```

SocketRead

Read data from a socket.

```
int  SocketRead(
   int           socket,              // socket
   uchar&        buffer[],            // buffer for reading data from socket
   uint          buffer_maxlen,       // number of bytes to read
   uint          timeout_ms           // reading timeout
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed to _LastError, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is activated.

buffer

[out] Reference to the uchar type array the data is read in. Dynamic array size is increased by the number of read bytes. The array size cannot exceed INT_MAX (2147483647).

buffer_maxlen

[in] Number of bytes to read to the *buffer[]* array. Data not fitting into the array remain in the socket. They can be received by the next SocketRead call. *buffer_maxlen* cannot exceed INT_MAX (2147483647).

timeout_ms

[in] Data reading timeout in milliseconds. If data is not obtained within this time, attempts are stopped and the function returns -1.

Return Value

If successful, return the number of read bytes. In case of an error, -1 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

In case of a data reading error, the error 5273 (ERR_NETSOCKET_IO_ERROR) is written in _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
```

```
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
         }
       else
         {
          Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
         }
       //--- close a socket after using
       SocketClose(socket);
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
   }
//+------------------------------------------------------------------+
```

See also

SocketTimeouts, MathSwap

SocketSend

Write data to a socket.

```
int  SocketSend(
   int           socket,              // socket
   const uchar&  buffer[],            // data buffer
   uint          buffer_len           // buffer size
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

buffer

[in] Reference to the uchar type array with the data to be sent to the socket.

buffer_len

[in] 'buffer' array size.

Return Value

If successful, return the number of bytes written to a socket. In case of an error, -1 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

In case of a data writing error, the error 5273 (ERR_NETSOCKET_IO_ERROR) is written to _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal setting
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
```

```
bool          ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- close a socket after using
      SocketClose(socket);
      }
    else
       Print("Failed to create a socket, error ",GetLastError());
   }
//+------------------------------------------------------------------+
```

See also

SocketTimeouts, MathSwap, StringToCharArray

SocketTlsHandshake

Initiate secure TLS (SSL) connection to a specified host via TLS Handshake protocol. During Handshake, a client and a server agree on connection parameters: applied protocol version and data encryption method.

```
bool  SocketTlsHandshake(
   int            socket,                 // socket
   const string   host                    // host address
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

host

[in] Address of a host a secure connection is established with.

Return Value

Returns true if successful, otherwise false.

Notes

Before a secure connection, the program should establish a standard TCP connection with the host using SocketConnect.

If secure connection fails, the error 5274 (ERR_NETSOCKET_HANDSHAKE_FAILED) is written to _LastError.

There is no need to call the function when connecting to the port 443. This is a standard TCP port used for secure TLS (SSL) connections.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                         SocketTlsHandshake.mq5 |
//|                              Copyright 2024, MetaQuotes Ltd. |
//|                                         https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   SERVER     "smtp.gmail.com"
#define   PORT       587
//+------------------------------------------------------------------+
//| Switch manually to secure connection                             |
//+------------------------------------------------------------------+
bool TlsHandshake(int socket)
  {
//--- get server greeting
   string rsp;

   if(!RecvString(socket,rsp))
      return(false);
//--- greet server
   if(!SendString(socket,"EHLO my.domain.com\r\n"))
      return(false);
//--- get a server response with a list of supported commands
   if(!RecvString(socket,rsp))
      return(false);
//--- print greeting
   Print("SERVER: ",rsp);
//--- inform the server that we want to switch from an insecure connection to a secure
   if(!SendString(socket,"STARTTLS\r\n"))
      return(false);
//--- get server response
   if(!RecvString(socket,rsp))
      return(false);
//--- in the example, we do not check the server response about readiness to switch to

//--- initiate a secure TLS (SSL) connection to the specified host using the TLS Hands
   if(SocketTlsHandshake(socket,InpTimeout))
      return(true);

   Print("SocketTlsHandshake() failed. Error ",GetLastError());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- create a socket and get its handle
   int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- connect to SERVER via PORT
   if(!SocketConnect(socket,SERVER,PORT,10000))
     {
      Print("SocketConnect() failed. Error ",GetLastError());
     }
   else
     {
      //--- insecure connection established
      PrintFormat("%s connection has been established to %s:%d",(PORT==443 ? "A secure
      //--- switch to secure connection
      if(PORT!=443 && TlsHandshake(socket))
        {
         PrintFormat("Unsecured connection to %s:%d switched to secured",SERVER,PORT);
         //--- if connection is protected by certificate, display its data
         string   subject,issuer,serial,thumbprint;
         datetime expiration;

         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:      ",subject);
            Print("   Issuer:     ",issuer);
            Print("   Number:     ",serial);
            Print("   Print:      ",thumbprint);
            Print("   Expiration: ",expiration);
           }
        }
     }
//--- close the socket after use
   SocketClose(socket);
   /*
   result:
   An unsecured connection has been established to smtp.gmail.com:587
   SERVER: 220 smtp.gmail.com ESMTP a640c23a62f3a-a9b1f298319sm82305866b.105 - gsmtp

   SERVER: 250-smtp.gmail.com at your service, [37.193.40.122]
   250-SIZE 35882577
   250-8BITMIME
   250-STARTTLS
   250-ENHANCEDSTATUSCODES
   250-PIPELINING
   250-CHUNKING
   250 SMTPUTF8

   SERVER: 220 2.0.0 Ready to start TLS

   SocketTlsHandshake(): A secure connection to smtp.gmail.com:587 is now established
   TLS certificate:
      Owner:  /CN=smtp.gmail.com
      Issuer:  /C=US/O=Google Trust Services/CN=WR2
      Number:     1f:f4:db:2a:5a:e6:dc:52:0a:4c:05:ce:81:cc:c3:f7
      Print: d6be8af229b5329cd3d4c2789c02aa94f89b421c
      Expiration: 2024.12.30 08:25:30
   */
```

```
  }
//+------------------------------------------------------------------+
//| Send string to server                                            |
//+------------------------------------------------------------------+
bool SendString(int socket,const string str)
  {
//--- convert the string to a character array
   uchar data[];
   int   size=StringToCharArray(str,data,0,str.Length(),CP_UTF8);
//--- send data to the socket
   ResetLastError();
   if(SocketSend(socket,data,size)==size)
      return(true);
//--- data sending error
   Print("Failed to send data to server. Error ",GetLastError());
   return false;
  }
//+------------------------------------------------------------------+
//| Get a string from the server                                     |
//+------------------------------------------------------------------+
bool RecvString(int socket,string& result,uint timeout_ms=1000)
  {
//--- wait for data to appear on the socket
   ulong wait_time_end=GetMicrosecondCount()+timeout_ms*1000;

   while(!SocketIsReadable(socket))
     {
      Sleep(10);
      //--- timed out waiting for data - return NULL as response
      if(wait_time_end<GetMicrosecondCount())
        {
         Print("ERROR: No response from server");
         return(false);
        }
     }
//--- read data from socket
   uchar  data[128];
   uint   size=0;
   string resp=NULL;

   do
     {
      uchar b[1];
      int   n=SocketRead(socket,b,1,1000);

      if(n < 0)
         break;

      if(n)
```

```
        {
         data[size++]=b[0];

         if(size==data.Size())
           {
            resp += CharArrayToString(data,0,data.Size(),CP_UTF8);
            size = 0;
           }
        }
     }
   while(SocketIsReadable(socket));
//--- copy the read data into the string
   if(size)
      resp+=CharArrayToString(data,0,size,CP_UTF8);
//--- if string is empty, then error
   if(!resp.Length())
     {
      Print("ERROR: No response from server");
      return(false);
     }
//--- return the string
   result=resp;
   return(true);
  }
```

SocketTlsCertificate

Get data on the certificate used to secure network connection.

```
bool  SocketTlsCertificate(
   int            socket,               // socket
   string&        subject,              // certificate owner
   string&        issuer,               // certificate issuer
   string&        serial,               // certificate serial number
   string&        thumbprint,           // certificate print
   datetime&      expiration            // certificate expiration
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

subject

[in] Certificate owner name. Corresponds to the Subject field.

issuer

[in] Certificate issuer name. Corresponds to the Issuer field.

serial

[in] Certificate serial number. Corresponds to the SerialNumber field.

thumbprint

[in] Certificate print. Corresponds to the SHA-1 hash from the entire certificate file (all fields including the issuer signature).

expiration

[in] Certificate expiration date in the datetime format.

Return Value

Returns true if successful, otherwise false.

Note

Certificate data can be requested only after establishing a secure connection using SocketTlsHandshake.

In case of a certificate obtaining error, the error 5275 (ERR_NETSOCKET_NO_CERTIFICATE) is written to _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
        {
         Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
        }
      //--- close a socket after using
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

SocketTlsRead

Read data from secure TLS connection.

```
int  SocketTlsRead(
   int            socket,               // socket
   uchar&         buffer[],             // buffer for reading data from socket
   uint           buffer_maxlen         // number of bytes to read
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed to _LastError, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is activated.

buffer

[out] Reference to the uchar type array the data is read in. Dynamic array size is increased by the number of read bytes. The array size cannot exceed INT_MAX (2147483647).

buffer_maxlen

[in] Number of bytes to read to the *buffer[]* array. Data not fitting into the array remain in the socket. They can be received by the next SocketTLSRead call. *buffer_maxlen* cannot exceed INT_MAX (2147483647).

Return Value

If successful, return the number of read bytes. In case of an error, -1 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

The function is executed till it receives the specified amount of data or the timeout is reached (SocketTimeouts).

In case of a data reading error, the error 5273 (ERR_NETSOCKET_IO_ERROR) is written in _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                                SocketExample.mq5 |
//|                        Copyright 2018, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "Copyright 2000-2024, MetaQuotes Ltd."
#property link        "https://www.mql5.com"
#property version     "1.00"
```

```
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =80;
bool         ExtTLS =false;
//+------------------------------------------------------------------+
//| Send command to the server                                       |
//+------------------------------------------------------------------+
bool HTTPSend(int socket,string request)
  {
   char req[];
   int  len=StringToCharArray(request,req)-1;
   if(len<0)
      return(false);
//--- if secure TLS connection is used via the port 443
   if(ExtTLS)
      return(SocketTlsSend(socket,req,len)==len);
//--- if standard TCP connection is used
   return(SocketSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout)
  {
   char   rsp[];
   string result;
   uint   timeout_check=GetTickCount()+timeout;
//--- read data from sockets till they are still present but not longer than timeout
   do
     {
      uint len=SocketIsReadable(socket);
      if(len)
        {
         int rsp_len;
         //--- various reading commands depending on whether the connection is secure
         if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
         else
            rsp_len=SocketRead(socket,rsp,len,timeout);
         //--- analyze the response
         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- print only the response header
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
           }
        }
     }
   while(GetTickCount()<timeout_check && !IsStopped());
   return(false);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- connect if all is well
      if(SocketConnect(socket,Address,Port,1000))
        {
         Print("Established connection to ",Address,":",Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is secured by the certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:  ",subject);
            Print("   Issuer:  ",issuer);
            Print("   Number:     ",serial);
            Print("   Print: ",thumbprint);
            Print("   Expiration: ",expiration);
            ExtTLS=true;
           }
         //--- send GET request to the server
         if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\
           {
            Print("GET request sent");
            //--- read the response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
```

```
         {
          Print("Connection to ",Address,":",Port," failed, error ",GetLastError());
         }
      //--- close a socket after using
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
```

See also

SocketTimeouts, MathSwap

SocketTlsReadAvailable

Read all available data from secure TLS connection.

```
int  SocketTlsReadAvailable(
   int            socket,              // socket
   uchar&         buffer[],            // buffer for reading data from socket
   const uint     buffer_maxlen        // number of bytes to read
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed to _LastError, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is activated.

buffer

[out] Reference to the uchar type array the data is read in. Dynamic array size is increased by the number of read bytes. The array size cannot exceed INT_MAX (2147483647).

buffer_maxlen

[in] Number of bytes to read to the buffer[] array. Data not fitting into the array remain in the socket. They can be received by the next SocketTlsReadAvailable or SocketTlsRead call. buffer_maxlen cannot exceed INT_MAX (2147483647).

Return Value

If successful, return the number of read bytes. In case of an error, -1 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

In case of a data reading error, the error 5273 (ERR_NETSOCKET_IO_ERROR) is written in _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                     SocketTlsReadAvailable.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version    "1.00"
#property script_show_inputs
//+------------------------------------------------------------------+
//| Script inputs                                                   |
//+-----------------------------------------------------------------+
input string InpMethod ="GET";            // Method (HEAD,GET)
input string InpServer ="www.google.com"; // Server
input uint   InpPort    =443;             // Port
input uint   InpTimeout=5000;             // Timeouts
//+-----------------------------------------------------------------+
//| Script program start function                                   |
//+-----------------------------------------------------------------+
void OnStart(void)
  {
   Print("Server: ",InpServer);
   Print("Port: ",InpPort);
//--- create a socket and get its handle
   const int socket=SocketCreate();

   if(socket==INVALID_HANDLE)
     {
      Print("SocketCreate() failed. Error ",GetLastError());
      return;
     }
//--- set timeouts for receiving and sending data for a socket system object
   if(!SocketTimeouts(socket,InpTimeout,InpTimeout))
     {
      PrintFormat("SocketTimeouts(%u, %u) failed. Error %d",InpTimeout,InpTimeout,GetL
      SocketClose(socket);
      return;
     }
//--- connect to Server via Port
   if(!SocketConnect(socket,InpServer,InpPort,InpTimeout))
     {
      PrintFormat("SocketConnect('%s', %u, %u) failed. Error %d",InpServer,InpPort,Inp
      SocketClose(socket);
      return;
     }
//--- get data on the certificate used to secure network connection
   string   subject,issuer,serial,thumbprint;
   datetime expiration;

   if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
     {
      Print("TLS certificate:");
      Print("   Owner:      ",subject);
      Print("   Issuer:     ",issuer);
      Print("   Number:     ",serial);
      Print("   Print:      ",thumbprint);
      Print("   Expiration: ",expiration);
     }
   else
```

```
     {
      //--- server does not provide a certificate - report an unsecured connection an
      Print("The connection is not secured by a certificate");
      SocketClose(socket);
      return;
     }
//--- send a request to the server
   string request=StringFormat("%s / HTTP/1.1\r\nHost: %s\r\nUser-Agent: MetaTrader 5

   if(HTTPSendTLS(socket,request))
     {
      //--- request sent - get response
      Print("\nRequest sent. Starting page loading...");
      uchar response[]; // all received data (document header and body)

      if(!HTTPRecvTLS(socket,response,InpTimeout))
        {
         Print("There were errors while reading the page");
         SocketClose(socket);
         return;
        }
      //--- report the number of bytes of data received
      PrintFormat("%u bytes received",response.Size());
      //--- display obtained page header only
      string result    =CharArrayToString(response,0,WHOLE_ARRAY,CP_UTF8);
      int    header_end=StringFind(result,"\r\n\r\n");

      if(header_end>0)
        {
         Print("\nHTTP answer header received:");
         Print(StringSubstr(result,0,header_end));
        }
     }
//--- close the socket after use
   SocketClose(socket);
   /*
   result:
   Server: www.google.com
   Port: 443
   TLS certificate:
      Owner: /CN=www.google.com
      Issuer: /C=US/O=Google Trust Services/CN=WR2
      Number: 0d:43:b1:4a:bb:9c:15:96:10:e1:3d:55:23:9f:25:4e
      Print: 89167618e5017f813aff981c88ce422dc1016bdf
      Expiration: 2024.12.30 08:26:35

   Request sent. Starting page loading...
   HTTPRecvTLS: Document received within 27 attempts
   25185 bytes received
   HTTP answer header received:
   HTTP/1.1 200 OK
   Date: Fri, 25 Oct 2024 17:12:42 GMT
   Expires: -1
   Cache-Control: private, max-age=0
   Content-Type: text/html; charset=ISO-8859-1
   Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src
   Accept-CH: Sec-CH-Prefers-Color-Scheme
   P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
   Server: gws
   X-XSS-Protection: 0
   X-Frame-Options: SAMEORIGIN
   Set-Cookie: AEC=AVYB7coyYMCdweTDTaWeGYzmRnxzKGqsOEosH_VkbCn8xhWkFz6v0kxQFw; expires
   Set-Cookie: NID=518=J02X02Ff4v_9sMcNoUz-1SolmuG08E26Gs438ik0J_SOJUMy7of-P-qup-LaNSW
   Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
   Accept-Ranges: none
   Vary: Accept-Encoding
   Transfer-Encoding: chunked
   */
  }
//+------------------------------------------------------------------+
//| Send an HTTP request over a secure connection                    |
//+------------------------------------------------------------------+
bool HTTPSendTLS(int socket,const string request)
  {
//--- convert the string to a character array, discard the terminating zero
   char req[];
   int  len=StringToCharArray(request,req,0,WHOLE_ARRAY,CP_UTF8)-1;

   if(len<0)
      return false;

   return(SocketTlsSend(socket,req,len)==len);
  }
//+------------------------------------------------------------------+
//| Get a web page via a secure connection                           |
//+------------------------------------------------------------------+
bool HTTPRecvTLS(int socket,uchar &response[],const uint timeout_ms)
  {
//--- read available data from secure TLS connection before the timeout expires
   ulong timeout_check=GetTickCount64()+timeout_ms;
   uchar block[1024];   // buffer for block data reading from socket
   uint  attempt=0;     // requested number of data blocks
   int   err    =0;     // error code

   ResetLastError();

   do
```

```
     {
      //--- read in blocks, maximum 1024 bytes
      int len=SocketTlsReadAvailable(socket,block,1024);

      if(len>0)
        {
         attempt++;
         //--- merge the obtained data blocks
         ArrayCopy(response,block,response.Size());
         //--- analyze the obtained data, define the header, page body, ending, or loa
         //...
         //...
         //...
         timeout_check=GetTickCount64()+timeout_ms;
        }
      else
         Sleep(10);

      err=GetLastError();
     }
   while(!IsStopped() && GetTickCount()<timeout_check && !err);
//--- were there any errors while reading?
   if(err)
     {
      Print("Error ",err);
      return(false);
     }

   PrintFormat("%s: Document received within %d attempts",__FUNCTION__,attempt);
   return(true);
  }
```

See also

SocketTimeouts, MathSwap

SocketTlsSend

Send data via secure TLS connection.

```
int  SocketTlsSend(
   int            socket,               // socket
   const uchar&  buffer[],              // data buffer
   uint           buffer_len            // buffer size
   );
```

Parameters

socket

[in] Socket handle returned by the SocketCreate function. When an incorrect handle is passed, the error 5270 (ERR_NETSOCKET_INVALIDHANDLE) is written to _LastError.

buffer

[in] Reference to the uchar type array with the data to be sent.

buffer_len

[in] 'buffer' array size.

Return Value

If successful, return the number of bytes written to a socket. In case of an error, -1 is returned.

Note

If an error occurs on a system socket when executing the function, connection established via SocketConnect is discontinued.

In case of a data writing error, the error 5273 (ERR_NETSOCKET_IO_ERROR) is written to _LastError.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If calling from an indicator, GetLastError() returns the error 4014 – "Function is not allowed for call".

Example:

```
//+------------------------------------------------------------------+
//|                                              SocketTlsSend.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version    "1.00"
#property description "Add Address to the list of allowed ones in the terminal settin
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port   =443;
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
   int socket=SocketCreate();
//--- check the handle
   if(socket!=INVALID_HANDLE)
     {
      //--- if all is well, connect
      if(SocketConnect(socket,Address,Port,1000))
        {
         PrintFormat("Established connection to %s:%d",Address,Port);

         string   subject,issuer,serial,thumbprint;
         datetime expiration;
         //--- if connection is protected by certificate, display its data
         if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
           {
            Print("TLS certificate:");
            Print("   Owner:      ",subject);
            Print("   Issuer:     ",issuer);
            Print("   Number:     ",serial);
            Print("   Print:      ",thumbprint);
            Print("   Expiration: ",expiration);
           }
         //--- send GET request to the server
         string request="GET / HTTP/1.1\r\nHost: www.mql5.com\r\nUser-Agent: MT5\r\n\
         char   req[];
         int    len=StringToCharArray(request,req)-1;

         if(len<0)
           {
            Print("StringToCharArray() failed. Error ", GetLastError());
            SocketClose(socket);
            return;
           }
         //--- if a secure TLS connection via port 443 is used
         if(SocketTlsSend(socket,req,len)==len)
           {
            Print("GET request sent");
            //--- read response
            if(!HTTPRecv(socket,1000))
               Print("Failed to get a response, error ",GetLastError());
           }
         else
            Print("Failed to send GET request, error ",GetLastError());
        }
      else
```

```
        {
         PrintFormat("Connection to %s:%d failed, error %d",Address,Port,GetLastError
        }
      //--- close the socket after use
      SocketClose(socket);
     }
   else
      Print("Failed to create a socket, error ",GetLastError());
  }
//+------------------------------------------------------------------+
//| Read server response                                             |
//+------------------------------------------------------------------+
bool HTTPRecv(int socket,uint timeout_ms)
  {
//--- read data from the socket while there is some, but not longer than timeout
   char   rsp[];
   string result;
   ulong  timeout_check=GetTickCount64()+timeout_ms;

   do
     {
      uint len=SocketIsReadable(socket);

      if(len)
        {
         //--- read and parse the data of a secure TLS connection
         int rsp_len=SocketTlsRead(socket,rsp,len);

         if(rsp_len>0)
           {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- display response header only
            int header_end=StringFind(result,"\r\n\r\n");

            if(header_end>0)
              {
               Print("HTTP answer header received:");
               Print(StringSubstr(result,0,header_end));
               return(true);
              }
            //--- update read timeout expiration time
            timeout_check=GetTickCount64()+timeout_ms;
           }
        }
     }
   while(GetTickCount64()<timeout_check && !IsStopped());

   return(false);
  }
```

See also

SocketTimeouts, MathSwap, StringToCharArray

WebRequest

The function sends an HTTP request to a specified server. The function has two versions:

1. **Sending simple requests** of type "key=value" using the header Content-Type: application/x-www-form-urlencoded.

```
int  WebRequest(
   const string      method,           // HTTP method
   const string      url,              // URL
   const string      cookie,           // cookie
   const string      referer,          // referer
   int               timeout,          // timeout
   const char        &data[],          // the array of the HTTP message body
   int               data_size,        // data[] array size in bytes
   char              &result[],        // an array containing server response data
   string            &result_headers   // headers of server response
   );
```

2. **Sending a request of any type** specifying the custom set of headers for a more flexible interaction with various Web services.

```
int  WebRequest(
   const string      method,           // HTTP method
   const string      url,              // URL
   const string      headers,          // headers
   int               timeout,          // timeout
   const char        &data[],          // the array of the HTTP message body
   char              &result[],        // an array containing server response data
   string            &result_headers   // headers of server response
   );
```

Parameters

method

[in] HTTP method.

url

[in] URL.

headers

[in] Request headers of type "key: value", separated by a line break "\r\n".

cookie

[in] Cookie value.

referer

[in] Value of the Referer header of the HTTP request.

timeout

[in] Timeout in milliseconds.

data[]

[in] Data array of the HTTP message body.

data_size

[in] Size of the data[] array.

result[]

[out] An array containing server response data.

result_headers

[out] Server response headers.

Return Value

HTTP server response code or -1 for an error.

Note

To use the WebRequest() function, add the addresses of the required servers in the list of allowed URLs in the "Expert Advisors" tab of the "Options" window. Server port is automatically selected on the basis of the specified protocol - 80 for "http://" and 443 for "https://".

The WebRequest() function is synchronous, which means its breaks the program execution and waits for the response from the requested server. Since the delays in receiving a response can be large, the function is not available for calls from indicators, because indicators run in a common thread shared by all indicators and charts on one symbol. Indicator performance delay on one of the charts of a symbol may stop updating of all charts of the same symbol.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If you try to call the function from an indicator, GetLastError() will return error 4014 – "Function is not allowed for call".

WebRequest() cannot be executed in the Strategy Tester.

Example:

```
void OnStart()
  {
   string cookie=NULL,headers;
   char   post[],result[];
   string url="https://finance.yahoo.com";
//--- To enable access to the server, you should add URL "https://finance.yahoo.com"
//--- to the list of allowed URLs (Main Menu->Tools->Options, tab "Expert Advisors"):
//--- Resetting the last error code
   ResetLastError();
//--- Downloading a html page from Yahoo Finance
   int res=WebRequest("GET",url,cookie,NULL,500,post,0,result,headers);
   if(res==-1)
     {
      Print("Error in WebRequest. Error code  =",GetLastError());
      //--- Perhaps the URL is not listed, display a message about the necessity to ac
      MessageBox("Add the address '"+url+"' to the list of allowed URLs on tab 'Expert
     }
```

```
 else
   {
    if(res==200)
      {
       //--- Successful download
       PrintFormat("The file has been successfully downloaded, File size %d byte.",
       //PrintFormat("Server headers: %s",headers);
       //--- Saving the data to a file
       int filehandle=FileOpen("url.htm",FILE_WRITE|FILE_BIN);
       if(filehandle!=INVALID_HANDLE)
         {
          //--- Saving the contents of the result[] array to a file
          FileWriteArray(filehandle,result,0,ArraySize(result));
          //--- Closing the file
          FileClose(filehandle);
         }
       else
          Print("Error in FileOpen. Error code =",GetLastError());
      }
    else
       PrintFormat("Downloading '%s' failed, error code %d",url,res);
   }
}
```

SendFTP

Sends a file at the address, specified in the setting window of the "FTP" tab.

```
bool  SendFTP(
   string  filename,          // file to be send by ftp
   string  ftp_path=NULL      // ftp catalog
   );
```

Parameters

filename

[in] Name of sent file.

ftp_path=NULL

[in] FTP catalog. If a directory is not specified, directory described in settings is used.

Return Value

In case of failure returns 'false'.

Note

Sent file must be located in the folder *terminal_directory\MQL5\files* or its subfolders. Sending isn't performed if FTP address and/or access password are not specified in settings.

SendFTP() function does not work in the Strategy Tester.

Example:

```
//+------------------------------------------------------------------+
//|                                                      SendFTP.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   FILENAME  "SomeFile.bin"
#define   PATH      NULL
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- check permission to send files to the FTP address in the terminal
   if(!TerminalInfoInteger(TERMINAL_FTP_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send messages to t
      return;
     }
//--- send file
   ResetLastError();
   if(!SendFTP(FILENAME, PATH))
      Print("SendFTP() failed. Error ",GetLastError());
  }
```

SendMail

Sends an email at the address specified in the settings window of the "Email" tab.

```
bool  SendMail(
   string  subject,       // header
   string  some_text      // email text
   );
```

Parameters

subject

[in] Email header.

some_text

[in] Email body.

Return Value

true - if an email is put into the send queue, otherwise - false.

Note

Sending can be prohibited in settings, email address can be omitted as well. For the error information call GetLastError().

SendMail() function does not work in the Strategy Tester.

Example:

```
//+------------------------------------------------------------------+
//|                                                     SendMail.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   SUBJECT   "Test SendMail"
#define   TEXT      "Text for SendMail() function"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- check permission to send email in the terminal
   if(!TerminalInfoInteger(TERMINAL_EMAIL_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send email message
      return;
     }
//--- send mail
   ResetLastError();
   if(!SendMail(SUBJECT, TEXT))
      Print("SendMail() failed. Error ",GetLastError());
  }
```

SendNotification

Sends push notifications to the mobile terminals, whose MetaQuotes IDs are specified in the "Notifications" tab.

```
bool  SendNotification(
   string  text           // Text of the notification
   );
```

Parameters

text

[in] The text of the notification. The message length should not exceed 255 characters.

Return Value

true if a notification has been successfully sent from the terminal; in case of failure returns false. When checking after a failed push of notification, GetLastError () may return one of the following errors:

- 4515 – ERR_NOTIFICATION_SEND_FAILED,
- 4516 – ERR_NOTIFICATION_WRONG_PARAMETER,
- 4517 – ERR_NOTIFICATION_WRONG_SETTINGS,
- 4518 – ERR_NOTIFICATION_TOO_FREQUENT.

Note

Strict use restrictions are set for the SendNotification() function: no more than 2 calls per second and not more than 10 calls per minute. Monitoring the frequency of use is dynamic. The function can be disabled in case of the restriction violation.

SendNotification() function does not work in the Strategy Tester.

Example:

```
//+------------------------------------------------------------------+
//|                                             SendNotification.mq5 |
//|                                  Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com
#property version     "1.00"

#define   MESSAGE   "Test Message"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart(void)
  {
//--- check permission to send notifications in the terminal
   if(!TerminalInfoInteger(TERMINAL_NOTIFICATIONS_ENABLED))
     {
      Print("Error. The client terminal does not have permission to send notifications
      return;
     }
//--- send notification
   ResetLastError();
   if(!SendNotification(MESSAGE))
      Print("SendNotification() failed. Error ",GetLastError());
  }
```

Global Variables of the Client Terminal

There is a group set of functions for working with global variables.

Global variables of the client terminal should not be mixed up with variables declared in the global scope of the mql5 program.

Global variables are kept in the client terminal for 4 weeks since the last access, then they will be deleted automatically. An access to a global variable is not only setting of a new value, but reading of the global variable value, as well.

Global variables of the client terminal are accessible simultaneously from all mql5 programs launched in the client terminal.

Function	Action
GlobalVariableCheck	Checks the existence of a global variable with the specified name
GlobalVariableTime	Returns time of the last accessing the global variable
GlobalVariableDel	Deletes a global variable
GlobalVariableGet	Returns the value of a global variable
GlobalVariableName	Returns the name of a global variable by its ordinal number in the list of global variables
GlobalVariableSet	Sets the new value to a global variable
GlobalVariablesFlush	Forcibly saves contents of all global variables to a disk
GlobalVariableTemp	Sets the new value to a global variable, that exists only in the current session of the terminal
GlobalVariableSetOnCondition	Sets the new value of the existing global variable by condition
GlobalVariablesDeleteAll	Deletes global variables with the specified prefix in their names
GlobalVariablesTotal	Returns the total number of global variables

GlobalVariableCheck

Checks the existence of a global variable with the specified name

```
bool  GlobalVariableCheck(
   string  name       // Global variable name
   );
```

Parameters

name

[in] Global variable name.

Return Value

Returns true, if the global variable exists, otherwise returns false.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

See also

GlobalVariableTime()

GlobalVariableTime

Returns the time when the global variable was last accessed.

```
datetime  GlobalVariableTime(
   string  name      // name
   );
```

Parameters

name

[in] Name of the global variable.

Return Value

The function returns time of last accessing the specified global variable. Addressing a variable for its value, for example using the GlobalVariableGet() and GlobalVariableCheck() functions, also modifies the time of last access. In order to obtain error details, call the GetLastError() function.

Note

Global variables exist in the client terminal during 4 weeks since they were called last. After that they are automatically deleted.

See also

GlobalVariableCheck()

GlobalVariableDel

Deletes a global variable from the client terminal.

```
bool  GlobalVariableDel(
   string  name       // Global variable name
   );
```

Parameters

name

[in] Global variable name.

Return Value

If successful, the function returns true, otherwise it returns false. To obtain an information about the error it is necessary to call the function GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableGet

Returns the value of an existing global variable of the client terminal. There are 2 variants of the function.

1. Immediately returns the value of the global variable.

```
double  GlobalVariableGet(
   string  name      // Global variable name
   );
```

2. Returns true or false depending on the success of the function run. If successful, the global variable of the client terminal is placed in a variable passed by reference in the second parameter.

```
bool  GlobalVariableGet(
   string  name,              // Global variable name
   double& double_var         // This variable will contain the value of the global va
   );
```

Parameters

name

[in] Global variable name.

double_var

[out] Target variable of the double type, which accepts the value stored in a the global variable of the client terminal.

Return Value

The value of the existing global variable or 0 in case of an error. For more details about the error, call GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableName

Returns the name of a global variable by its ordinal number.

```
string  GlobalVariableName(
   int  index      // Global variable number in the list of global variables
   );
```

Parameters

index

[in] Sequence number in the list of global variables. It should be greater than or equal to 0 and less than GlobalVariablesTotal().

Return Value

Global variable name by its ordinal number in the list of global variables. For more details about the error, call GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableSet

Sets a new value for a global variable. If the variable does not exist, the system creates a new global variable.

```
datetime  GlobalVariableSet(
   string  name,       // Global variable name
   double  value       // Value to set
   );
```

Parameters

name

[in] Global variable name.

value

[in] The new numerical value.

Return Value

If successful, the function returns the last modification time, otherwise 0. For more details about the error, call GetLastError().

Note

A global variable name should not exceed 63 characters. Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesFlush

Forcibly saves contents of all global variables to a disk.

```
void  GlobalVariablesFlush();
```

Return Value

No return value.

Note

The terminal writes all the global variables when the work is over, but data can be lost at a sudden computer operation failure. This function allows independently controlling the process of saving global variables in case of contingency.

GlobalVariableTemp

The function attempts to create a temporary global variable. If the variable doesn't exist, the system creates a new temporary global variable.

```
bool  GlobalVariableTemp(
   string  name      // Global variable name
   );
```

Parameters

name

[in] The name of a temporary global variable.

Return Value

If successful, the function returns true, otherwise - false. To get details about the error, you should call the GetLastError() function.

Note

Temporary global variables exist only while the client terminal is running; after the terminal shutdown they are automatically deleted. Note that during the execution of GlobalVariablesFlush() temporary global variables are not written to a disk.

After a temporary global variable has been created, it can be accessed and modified the same as global variable of the client terminal.

GlobalVariableSetOnCondition

Sets the new value of the existing global variable if the current value equals to the third parameter check_value. If there is no global variable, the function will generate an error ERR_GLOBALVARIABLE_NOT_FOUND (4501) and return false.

```
bool  GlobalVariableSetOnCondition(
   string  name,            // Global variable name
   double  value,           // New value for variable if condition is true
   double  check_value      // Check value condition
   );
```

Parameters

name

[in] The name of a global variable.

value

[in] New value.

check_value

[in] The value to check the current value of the global variable.

Return Value

If successful, the function returns true, otherwise it returns false. For details about the error call GetLastError(). If the current value of the global variable is different from check_value, the function returns false.

Note

Function provides atomic access to the global variable, so it can be used for providing of a mutex at interaction of several Expert Advisors working simultaneously within one client terminal.

GlobalVariablesDeleteAll

Deletes global variables of the client terminal.

```
int  GlobalVariablesDeleteAll(
   string     prefix_name=NULL,     // All global variables with names beginning with
   datetime    limit_data=0          // All global variables that were changed before t
   );
```

Parameters

prefix_name=NULL

[in] Name prefix global variables to remove. If you specify a prefix NULL or empty string, then all variables that meet the data criterion will be deleted.

limit_data=0

[in] Date to select global variables by the time of their last modification. The function removes global variables, which were changed before this date. If the parameter is zero, then all variables that meet the first criterion (prefix) are deleted.

Return Value

The number of deleted variables.

Note

If both options are equal to zero (prefix_name = NULL and limit_data = 0), then function deletes all global variables of the terminal. If both parameters are specified, then it deletes global variables corresponding to both parameters.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesTotal

Returns the total number of global variables of the client terminal.

```
int  GlobalVariablesTotal();
```

Return Value

Number of global variables.

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted. Call of a global variable is not only setting a new value, but also reading the value of the global variable.

File Functions

This is a group of functions for working with files.

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means cannot be outside the file sandbox.

There are two directories (with subdirectories) in which working files can be located:

- terminal_data_folder\MQL5\FILES\ (in the terminal menu select to view "File" - "Open the data directory");
- the common folder for all the terminals installed on a computer - usually located in the directory C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\Files.

There is a program method to obtain names of these catalogs using the TerminalInfoString() function, using the ENUM_TERMINAL_INFO_STRING enumeration:

```
//--- Folder that stores the terminal data
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
//--- Common folder for all client terminals
   string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

Work with files from other directories is prohibited.

If the file is opened for writing using FileOpen(), all subfolders specified in the path will be created if there are no such ones.

File functions allow working with so-called "named pipes". To do this, simply call FileOpen() function with appropriate parameters.

Function	Action
FileSelectDialog	Create a file or folder opening/creation dialog
FileFindFirst	Starts the search of files in a directory in accordance with the specified filter
FileFindNext	Continues the search started by the FileFindFirst() function
FileFindClose	Closes search handle
FileOpen	Opens a file with a specified name and flag
FileDelete	Deletes a specified file
FileFlush	Writes to a disk all data remaining in the input/output file buffer
FileGetInteger	Gets an integer property of a file
FileIsEnding	Defines the end of a file in the process of reading
FileIsLineEnding	Defines the end of a line in a text file in the process of reading
FileClose	Closes a previously opened file
FileIsExist	Checks the existence of a file

Function	Action
FileCopy	Copies the original file from a local or shared folder to another file
FileMove	Moves or renames a file
FileReadArray	Reads arrays of any type except for string from the file of the BIN type
FileReadBool	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type
FileReadDatetime	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value
FileReadDouble	Reads a double value from the current position of the file pointer
FileReadFloat	Reads a float value from the current position of the file pointer
FileReadInteger	Reads int, short or char value from the current position of the file pointer
FileReadLong	Reads a long type value from the current position of the file pointer
FileReadNumber	Reads from the file of the CSV type a string from the current position till a delimiter (or til the end of a text line) and converts the read string into double value
FileReadString	Reads a string from the current position of a file pointer from a file
FileReadStruct	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer
FileSeek	Moves the position of the file pointer by a specified number of bytes relative to the specified position
FileSize	Returns the size of a corresponding open file
FileTell	Returns the current position of the file pointer of a corresponding open file
FileWrite	Writes data to a file of CSV or TXT type
FileWriteArray	Writes arrays of any type except for string into a file of BIN type
FileWriteDouble	Writes value of the double type from the current position of a file pointer into a binary file
FileWriteFloat	Writes value of the float type from the current position of a file pointer into a binary file
FileWriteInteger	Writes value of the int type from the current position of a file pointer into a binary file
FileWriteLong	Writes value of the long type from the current position of a file pointer into a binary file

Function	Action
FileWriteString	Writes the value of a string parameter into a BIN or TXT file starting from the current position of the file pointer
FileWriteStruct	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer
FileLoad	Reads all data of a specified binary file into a passed array of numeric types or simple structures
FileSave	Writes to a binary file all elements of an array passed as a parameter
FolderCreate	Creates a folder in the Files directory
FolderDelete	Removes a selected directory. If the folder is not empty, then it can't be removed
FolderClean	Deletes all files in the specified folder

FileSelectDialog

Create a file or folder opening/creation dialog.

```
int  FileSelectDialog(
   string    caption,              // window header
   string    initial_dir,          // initial directory
   string    filter,               // extension filter
   uint      flags,                // combination of flags
   string&   filenames[],          // array with file names
   string    default_filename      // default file name
   );
```

Parameters

caption

[in] Dialog window header.

initial_dir

[in] Initial directory name relative to MQL5\Files, the contents of which is to be displayed in the dialog box. If the value is NULL, MQL5\Files is displayed in the dialog.

filter

[in] Extension filter of the files to be displayed in the selection dialog window. Files of other formats are hidden.

flags

[in] Combination of flags defining the dialog window mode. The flags are defined as follows:
FSD_WRITE_FILE – file open dialog;
FSD_SELECT_FOLDER – allow selection of folders only;
FSD_ALLOW_MULTISELECT – allow selection of multiple files;
FSD_FILE_MUST_EXIST – selected files should exist;
FSD_COMMON_FOLDER – file is located in the common folder of all client terminals \Terminal\Common\Files.

filenames[]

[out] Array of strings the names of selected files/folders are placed to.

default_filename

[in] Default file/folder name. If specified, a name is automatically added to the open dialog and returned in the *filenames[]* array when testing.

Return Value

In case of a successful completion, the function returns the number of selected files whose names can be obtained in *filenames[]*. If a user closes the dialog without selecting a file, the function returns 0. In case of unsuccessful execution, a value less than 0 is returned. The error code can be obtained using GetLastError().

Note

For reasons of security, working with files is strictly controlled in MQL5 language. Files used in file operations by means of MQL5 language cannot be located outside the file sandbox (namely, outside the MQL5\Files directory).

The *initial_dir* name is searched in the client terminal's directory in MQL5\Files (or testing_agent_directory\MQL5\Files in case of testing). If FSD_COMMON_FOLDER is set among the flags, the search for the initial directory is performed in the common folder of all client terminals \Terminal\Common\Files.

The *filter* parameter indicates valid files and should be set in the "<description 1>|<extension 1>|<description 2>|<extension 2>..." format, for example, "Text files (*.txt)|*.txt|All files (*.*)|*.*". The first extension "Text files (*.txt)|*.txt" is selected as a default file type.
If *filter*=NULL, the mask of file selection in the dialog window is "All Files (*.*)|*.*|"

If the *default_filename* parameter is set, calling FileSelectDialog() returns 1, while the *default_filename* value itself is copied to the *filenames[]* array during the non-visualized testing.

The function cannot be used in custom indicators since calling FileSelectDialog() suspends the thread of execution for the whole time while waiting for the user's response. Since all indicators for each symbol are executed in a single thread, such suspension makes the operation of all charts on all timeframes for this symbol impossible.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get the names of text files for downloading from the common folder of the client
   string filenames[];
   if(FileSelectDialog("Select files to download", NULL,
                       "Text files (*.txt)|*.txt|All files (*.*)|*.*",
                       FSD_ALLOW_MULTISELECT|FSD_COMMON_FOLDER, filenames, "data.txt")
     {
      //--- display the name of each selected file
      int total=ArraySize(filenames);
      for(int i=0; i<total; i++)
         Print(i, ": ", filenames[i]);
     }
   else
     {
     Print("Files not selected");
     }
//---
  }
```

See also

FileOpen, FileIsExist, FileDelete, FileMove, FolderCreate, FolderDelete, FolderClean, Flags of opening files

FileFindFirst

The function starts the search of files or subdirectories in a directory in accordance with the specified filter.

```
long  FileFindFirst(
   const string   file_filter,          // String - search filter
   string&        returned_filename,    // Name of the file or subdirectory found
   int            common_flag=0         // Defines the search
   );
```

Parameters

file_filter

[in] Search filter. A subdirectory (or sequence of nested subdirectories) relative to the \Files directory, in which files should be searched for, can be specified in the filter.

returned_filename

[out] The returned parameter, where, in case of success, the name of the first found file or subdirectory is placed. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

Returns handle of the object searched, which should be used for further sorting of files and subdirectories by the FileFindNext() function, or INVALID_HANDLE when there is no file and subdirectory corresponding to the filter (in the particular case - when the directory is empty). After searching, the handle must be closed using the FileFindClose() function.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="Dir1\\*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   string int_dir="";
   int    i=1,pos=0,last_pos=-1;
//--- search for the last backslash
   while(!IsStopped())
     {
      pos=StringFind(InpFilter,"\\",pos+1);
      if(pos>=0)
         last_pos=pos;
      else
         break;
     }
//--- the filter contains the folder name
   if(last_pos>=0)
      int_dir=StringSubstr(InpFilter,0,last_pos+1);
//--- get the search handle in the root of the local folder
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if the FileFindFirst() is executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- in a loop, check if the passed strings are the names of files or director
      do
        {
         ResetLastError();
         //--- if it's a file, the function returns true, and if it's a directory, it
         FileIsExist(int_dir+file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "D
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

[FileFindNext](), [FileFindClose]()

FileFindNext

The function continues the search started by FileFindFirst().

```
bool  FileFindNext(
   long      search_handle,          // Search handle
   string&   returned_filename       // Name of the file or subdirectory found
   );
```

Parameters

search_handle

[in] Search handle, retrieved by FileFindFirst().

returned_filename

[out] The name of the next file or subdirectory found. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

Return Value

If successful returns true, otherwise false.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- receive search handle in local folder's root
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if FileFindFirst() function executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- check if the passed strings are file or directory names in the loop
      do
        {
         ResetLastError();
         //--- if this is a file, the function will return true, if it is a directory,
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "Di
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

[FileFindFirst](), [FileFindClose]()

FileFindClose

The function closes the search handle.

```
void  FileFindClose(
   long  search_handle      //  Search handle
   );
```

Parameters

search_handle

[in] Search handle, retrieved by FileFindFirst().

Return Value

No value returned.

Note

Function must be called to free up system resources.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- receive search handle in local folder's root
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if FileFindFirst() function executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- check if the passed strings are file or directory names in the loop
      do
        {
         ResetLastError();
         //--- if this is a file, the function will return true, if it is a directory,
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File"
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

[FileFindFirst](), [FileFindNext]()

FileIsExist

Checks the existence of a file.

```
bool  FileIsExist(
   const string  file_name,        // File name
   int           common_flag=0     // Search area
   );
```

Parameters

file_name

[in] The name of the file being checked

common_flag=0

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

Returns true, if the specified file exists.

Note

Checked file can turn out to be a subdirectory. In this case, FileIsExist() function will return false, while error 5018 will be logged in _LastError variable - "This is a directory, not a file" (see example for FileFindFirst function).

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If common_flag = FILE_COMMON, then the function looks for the file in a shared folder for all client terminals \Terminal\Common\Files, otherwise the function looks for a file in a local folder (MQL5\Files or MQL5\Tester\Files in the case of testing).

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.txt"; // filter for searching the files
   datetime create_date;    // file creation date
   string   files[];        // list of file names
   int      def_size=25;    // array size by default
   int      size=0;         // number of files
//--- allocate memory for the array
   ArrayResize(files,def_size);
//--- receive the search handle in the local folder's root
   long search_handle=FileFindFirst(filter,file_name);
//--- check if FileFindFirst() executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- searching files in the loop
      do
        {
         files[size]=file_name;
         //--- increase the array size
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reset the error value
         ResetLastError();
         //--- receive the file creation date
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- check if the file is old
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- check what files have remained
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
         PrintFormat("%s file deleted!",files[i]);
     }
  }
```

See also

FileFindFirst

FileOpen

The function opens the file with the specified name and flag.

```
int  FileOpen(
   string  file_name,           // File name
   int     open_flags,          // Combination of flags
   short   delimiter='\t',      // Delimiter
   uint    codepage=CP_ACP      // Code page
   );
```

Parameters

file_name

[in] The name of the file can contain subfolders. If the file is opened for writing, these subfolders will be created if there are no such ones.

open_flags

[in] combination of flags determining the operation mode for the file. The flags are defined as follows:
FILE_READ file is opened for reading
FILE_WRITE file is opened for writing
FILE_BIN binary read-write mode (no conversion from a string and to a string)
FILE_CSV file of csv type (all recorded items are converted to the strings of unicode or ansi type, and are separated by a delimiter)
FILE_TXT a simple text file (the same as csv, but the delimiter is not taken into account)
FILE_ANSI lines of ANSI type (single-byte symbols)
FILE_UNICODE lines of UNICODE type (double-byte characters)
FILE_SHARE_READ shared reading from several programs
FILE_SHARE_WRITE shared writing from several programs
FILE_COMMON location of the file in a shared folder for all client terminals \Terminal\Common\Files

delimiter='\t'

[in] value to be used as a separator in txt or csv-file. If the csv-file delimiter is not specified, it defaults to a tab. If the txt-file delimiter is not specified, then no separator is used. If the separator is clearly set to 0, then no separator is used.

codepage=CP_ACP

[in] The value of the code page. For the most-used code pages provide appropriate constants.

Return Value

If a file has been opened successfully, the function returns the file handle, which is then used to access the file data. In case of failure returns INVALID_HANDLE.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Make sure to set the FILE_ANSI flag if the file should be read in a specific encoding (the codepage parameter with a code page value is specified). If there is no specified FILE_ANSI flag, the text file is read in Unicode without any conversion.

The file is opened in the folder of the client terminal in the subfolder MQL5\files (or testing_agent_directory\MQL5\files in case of testing). If FILE_COMMON is specified among flags, the file is opened in a shared folder for all MetaTrader 5 client terminals.

"Named pipes" can be opened according to the following rules:

- Pipe name is a string, which should have the following look: "\\servername\pipe\pipename", where servername - server name in the network, while pipename is a pipe name. If the pipes are used on the same computer, the server name can be omitted but a point should be inserted instead of it: "\\.\pipe\pipename". A client trying to connect the pipe should know its name.
- FileFlush() and FileSeek() should be called to the beginning of a file between sequential operations of reading from the pipe and writing to it.

A special symbol '\' is used in shown strings. Therefore, '\' should be doubled when writing a name in MQL5 application. It means that the above example should have the following look in the code: "\\\\servername\\pipe\\pipename".

More information about working with named pipes can be found in the article "Communicating With MetaTrader 5 Using Named Pipes Without Using DLLs".

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- incorrect file opening method
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   string filename=terminal_data_path+"\\MQL5\\Files\\"+"fractals.csv";
   int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
   if(filehandle<0)
     {
      Print("Failed to open the file by the absolute path ");
      Print("Error code ",GetLastError());
     }

//--- correct way of working in the "file sandbox"
   ResetLastError();
   filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
   if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(_Period));
      FileClose(filehandle);
      Print("FileOpen OK");
     }
   else Print("Operation FileOpen failed, error ",GetLastError());
//--- another example with the creation of an enclosed directory in MQL5\Files\
    string subfolder="Research";
    filehandle=FileOpen(subfolder+"\\fractals.txt",FILE_WRITE|FILE_CSV);
       if(filehandle!=INVALID_HANDLE)
      {
       FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(_Period));
       FileClose(filehandle);
       Print("The file must be created in the folder "+terminal_data_path+"\\"+subfolde
      }
    else Print("File open failed, error ",GetLastError());
   }
```

See also

Use of a Codepage, FileFindFirst, FolderCreate, File opening flags

FileClose

Close the file previously opened by FileOpen().

```
void  FileClose(
   int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

No value returned.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the path to the file we are going to use
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
//--- reset the error value
   ResetLastError();
//--- open the file for reading (if the file does not exist, the error will occur)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      //--- print the file contents
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

FileCopy

The function copies the original file from a local or shared folder to another file.

```
bool  FileCopy(
   const string  src_file_name,      // Name of a source file
   int           common_flag,        // Location
   const string  dst_file_name,      // Name of the destination file
   int           mode_flags          // Access mode
   );
```

Parameters

src_file_name

[in] File name to copy.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder (for example, *common_flag=0*).

dst_file_name

[in] Result file name.

mode_flags

[in] Access flags. The parameter can contain only 2 flags: FILE_REWRITE and/or FILE_COMMON - other flags are ignored. If the file already exists, and the FILE_REWRITE flag hasn't been specified, then the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the FILE_REWRITE flag in the mode_flags parameter.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpSrc="source.txt";       // source
input string InpDst="destination.txt";  // copy
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- display the source contents (it must exist)
   if(!FileDisplay(InpSrc))
      return;
//--- check if the copy file already exists (may not be created)
   if(!FileDisplay(InpDst))
     {
      //--- the copy file does not exist, copying without FILE_REWRITE flag (correct c
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
   else
     {
      //--- the copy file already exists, try to copy without FILE_REWRITE flag (incor
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
      //--- InpDst file's contents remains the same
      FileDisplay(InpDst);
      //--- copy once more with FILE_REWRITE flag (correct copying if the file exists)
      if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
//--- receive InpSrc file copy
   FileDisplay(InpDst);
  }
//+------------------------------------------------------------------+
//| Read the file contents                                           |
//+------------------------------------------------------------------+
bool FileDisplay(const string file_name)
  {
//--- reset the error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- display the file contents in the loop
      Print("+---------------------+");
      PrintFormat("File name = %s",file_name);
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      Print("+---------------------+");
      //--- close the file
      FileClose(file_handle);
      return(true);
     }
//--- failed to open the file
   PrintFormat("%s is not opened, error = %d",file_name,GetLastError());
   return(false);
  }
```

FileDelete

Deletes the specified file in a local folder of the client terminal.

```
bool  FileDelete(
   const string  file_name,      // File name to delete
   int           common_flag=0   // Location of the file to delete
   );
```

Parameters

file_name

[in] File name.

common_flag=0

[in] Flag determining the file location. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

Deletes the specified file from a local folder of the client terminal (MQL5\Files or MQL5\Tester\Files in case of testing). If common_flag = FILE_COMMON, then the function removes the file from the shared folder for all client terminals \Terminal\Common\Files.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.txt"; // filter for searching the files
   datetime create_date;    // file creation date
   string   files[];        // list of file names
   int      def_size=25;    // array size by default
   int      size=0;         // number of files
//--- allocate memory for the array
   ArrayResize(files,def_size);
//--- receive the search handle in the local folder's root
   long search_handle=FileFindFirst(filter,file_name);
//--- check if FileFindFirst() executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- searching files in the loop
      do
        {
         files[size]=file_name;
         //--- increase the array size
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reset the error value
         ResetLastError();
         //--- receive the file creation date
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- check if the file is old
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- check what files have remained
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
         PrintFormat("%s file deleted!",files[i]);
     }
  }
```

FileMove

Moves a file from a local or shared folder to another folder.

```
bool  FileMove(
   const string  src_file_name,     // File name for the move operation
   int           common_flag,       // Location
   const string  dst_file_name,     // Name of the destination file
   int           mode_flags         // Access mode
   );
```

Parameters

src_file_name

[in] File name to move/rename.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder (*common_flag=0*).

dst_file_name

[in] File name after operation

mode_flags

[in] Access flags. The parameter can contain only 2 flags: FILE_REWRITE and/or FILE_COMMON - other flags are ignored. If the file already exists and the FILE_REWRITE flag isn't specified, the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the FILE_REWRITE flag in the mode_flags parameter.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpSrcName="data.txt";
input string InpDstName="newdata.txt";
input string InpSrcDirectory="SomeFolder";
input string InpDstDirectory="OtherFolder";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string local=TerminalInfoString(TERMINAL_DATA_PATH);
   string common=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- receive file paths
   string src_path;
   string dst_path;
   StringConcatenate(src_path,InpSrcDirectory,"//",InpSrcName);
   StringConcatenate(dst_path,InpDstDirectory,"//",InpDstName);
//--- check if the source file exists (if not - exit)
   if(FileIsExist(src_path))
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpSrcName,local,InpSrc
   else
     {
      PrintFormat("Error, %s source file not found",InpSrcName);
      return;
     }
//--- check if the result file already exists
   if(FileIsExist(dst_path,FILE_COMMON))
     {
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpDstName,common,InpDs
      //--- file exists, moving should be performed with FILE_REWRITE flag
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON|FILE_REWRITE))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
   else
     {
      PrintFormat("%s file does not exist in the %s\\Files\\%s folder",InpDstName,comm
      //--- the file does not exist, moving should be performed without FILE_REWRITE
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
//--- the file is moved; let's check it out
   if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
      Print("Success!");
   else
      Print("Error!");
  }
```

See also

FileIsExist

FileFlush

Writes to a disk all data remaining in the input/output file buffer.

```
void  FileFlush(
   int   file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

No value returned.

Note

When writing to a file, the data may be actually found there only after some time. To save the data in the file instantly, use FileFlush() function. If the function is not used, part of the data that has not been stored in the disk yet, will be forcibly written there only when the file is closed using FileClose() function.

The function should be used when written data is of a certain value. It should be kept in mind that frequent function call may affect the program operation speed.

Function FileFlush () must be called between the operations of reading from a file and writing to it.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- file name for writing
input string InpFileName="example.csv"; // file name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- reset error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- write data to the file
      for(int i=0;i<1000;i++)
        {
         //--- call write function
         FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYMBOL_BID),Sy
         //--- save data on the disk at each 128th iteration
         if((i & 127)==127)
```

```
           {
            //--- now, data will be located in the file and will not be lost in case
            FileFlush(file_handle);
            PrintFormat("i = %d, OK",i);
           }
         //--- 0.01 second pause
         Sleep(10);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

See also

FileClose

FileGetInteger

Gets an integer property of a file. There are two variants of the function.

1. Get a property by the handle of a file.

```
long  FileGetInteger(
   int                         file_handle,    // File handle
   ENUM_FILE_PROPERTY_INTEGER  property_id     // Property ID
   );
```

2. Get a property by the file name.

```
long  FileGetInteger(
   const string                file_name,             // File name
   ENUM_FILE_PROPERTY_INTEGER  property_id,           // Property ID
   bool                        common_folder=false    // The file is viewed in a local
   );                                                 // or a common folder of all terr
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

file_name

[in] File name.

property_id

[in] File property ID. The value can be one of the values of the ENUM_FILE_PROPERTY_INTEGER enumeration. If the second variant of the function is used, you can receive only the values of the following properties: FILE_EXISTS, FILE_CREATE_DATE, FILE_MODIFY_DATE, FILE_ACCESS_DATE and FILE_SIZE.

common_folder=false

[in] Points to the file location. If the parameter is false, terminal data folder is viewed. Otherwise it is assumed that the file is in the shared folder of all terminals \Terminal\Common\Files (FILE_COMMON).

Return Value

The value of the property. In case of an error, -1 is returned. To get an error code use the GetLastError() function.

If a folder is specified when getting properties by the name, the function will have error 5018 (ERR_MQL_FILE_IS_DIRECTORY) in any case, though the return value will be correct.

Note

The function always changes the error code. In case of successful completion the error code is reset to NULL.

Example:

```
//--- display the window of input parameters when launching the script
```

```
#property script_show_inputs
//--- input parameters
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string path=InpDirectoryName+"//"+InpFileName;
   long   l=0;
//--- open the file
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_CSV);
   if(handle!=INVALID_HANDLE)
     {
      //--- print all information about the file
      Print(InpFileName," file info:");
      FileInfo(handle,FILE_EXISTS,l,"bool");
      FileInfo(handle,FILE_CREATE_DATE,l,"date");
      FileInfo(handle,FILE_MODIFY_DATE,l,"date");
      FileInfo(handle,FILE_ACCESS_DATE,l,"date");
      FileInfo(handle,FILE_SIZE,l,"other");
      FileInfo(handle,FILE_POSITION,l,"other");
      FileInfo(handle,FILE_END,l,"bool");
      FileInfo(handle,FILE_IS_COMMON,l,"bool");
      FileInfo(handle,FILE_IS_TEXT,l,"bool");
      FileInfo(handle,FILE_IS_BINARY,l,"bool");
      FileInfo(handle,FILE_IS_CSV,l,"bool");
      FileInfo(handle,FILE_IS_ANSI,l,"bool");
      FileInfo(handle,FILE_IS_READABLE,l,"bool");
      FileInfo(handle,FILE_IS_WRITABLE,l,"bool");
      //--- close the file
      FileClose(handle);
     }
   else
      PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetLastError());
  }
//+------------------------------------------------------------------+
//| Display the value of the file property                           |
//+------------------------------------------------------------------+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
  {
//--- receive the property value
   ResetLastError();
   if((l=FileGetInteger(handle,id))!=-1)
     {
      //--- the value received, display it in the correct format
      if(!StringCompare(type,"bool"))
         Print(EnumToString(id)," = ",l ? "true" : "false");
      if(!StringCompare(type,"date"))
         Print(EnumToString(id)," = ",(datetime)l);
      if(!StringCompare(type,"other"))
         Print(EnumToString(id)," = ",l);
     }
   else
      Print("Error, Code = ",GetLastError());
  }
```

See also

File Operations, File Properties

FileIsEnding

Defines the end of a file in the process of reading.

```
bool  FileIsEnding(
   int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The function returns true if the file end has been reached in the process of reading or moving of the file pointer.

Note

To define the end of the file, the function tries to read the next string from it. If the string does not exist, the function returns true, otherwise it returns false.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the path to the file we are going to use
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DATA_PATH));
//--- reset the error value
   ResetLastError();
//--- open the file for reading (if the file does not exist, the error will occur)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpE
   if(file_handle!=INVALID_HANDLE)
     {
      //--- print the file contents
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

FileIsLineEnding

Defines the line end in a text file in the process of reading.

```
bool  FileIsLineEnding(
   int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Returns true if in the process of reading txt or csv-file reached the end of the line (the characters CR-LF).

Example (the file obtained during the execution of an example for FileWriteString function is used here)

```
//+------------------------------------------------------------------+
//|                                        Demo_FileIsLineEnding.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="RSI.csv";   // file name
input string InpDirectoryName="Data"; // directory name
//--- indicator buffers
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- overbought variables
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
```

```
//--- oversold variables
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- variables of array sizes by default
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- allocate memory for arrays
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      double value;
      //--- read data from file
      while(!FileIsEnding(file_handle))
        {
         //--- read the first value in the string
         value=FileReadNumber(file_handle);
         //--- read to different arrays according to the function result
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
```

```
//---- set the indicator values that will not be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Read the file's string data                                      |
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- read till the end of the string or of the file is reached
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- shift the position by reading the number
      if(flag)
         FileReadNumber(file_handle);
      //--- store the current date
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- increase the array size if necessary
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- slip past the first iteration
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
```

```
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- check if any data is still present
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- if the dates coincide, the bar is in the overbought area
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - red color
               color_buff[i]=0;
               //--- increase the counter
               ovb_ind=j+1;
               break;
              }
           }
      //--- check if any data still exists
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- if the dates coincide, the bar is in the oversold area
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - blue color
               color_buff[i]=1;
               //--- increase the counter
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| ChartEvent event handler                                         |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
                  )
  {
//--- change the indicator width according to the scale
   if(ChartGetInteger(0,CHART_SCALE)>3)
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
   else
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
  }
```

See also

FileWriteString

FileReadArray

Reads from a file of BIN type arrays of any type except string (may be an array of structures, not containing strings, and dynamic arrays).

```
uint  FileReadArray(
   int     file_handle,               // File handle
   void&   array[],                   // Array to record
   int     start=0,                   // start array position to write
   int     count=WHOLE_ARRAY          // count to read
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

array[]

[out] An array where the data will be loaded.

start=0

[in] Start position to read from the array.

count=WHOLE_ARRAY

[in] Number of elements to read. By default, reads the entire array (count=WHOLE_ARRAY).

Return Value

Number of elements read.

Note

String array can be read only from the file of TXT type. If necessary, the function tries to increase the size of the array.

Example (the file obtained after execution of the example for FileWriteArray function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Structure for storing price data                                 |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // date
   double            bid;  // bid price
   double            ask;  // ask price
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- structure array
   prices arr[];
//--- file path
   string path=InpDirectoryName+"//"+InpFileName;
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- read all data from the file to the array
      FileReadArray(file_handle,arr);
      //--- receive the array size
      int size=ArraySize(arr);
      //--- print data from the array
      for(int i=0;i<size;i++)
         Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i].ask);
      Print("Total data = ",size);
      //--- close the file
      FileClose(file_handle);
     }
   else
      Print("File open failed, error ",GetLastError());
  }
```

See also

Variables, FileWriteArray

FileReadBool

Reads from the file of CSV type string from the current position to a delimiter (or till the end of the text line) and converts the read string to a bool type value.

```
bool  FileReadBool(
   int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Line read may be set to "true", "false" or the symbolic representation of integers "0" or "1". A nonzero value is converted to a logical true. The function returns the converted value.

Example (the file obtained after executing the example for FileWrite function is used here)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadBool.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv";  // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;       // index
double   upbuff[];    // indicator buffers of up arrows
double   downbuff[];  // indicator buffer of down arrows
```

```
bool      sign_buff[]; // signal array (true - buy, false - sell)
datetime time_buff[]; // array of signals' arrival time
int       size=0;      // size of signal arrays
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- first, read the number of signals
      size=(int)FileReadNumber(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         //--- signal time
         time_buff[i]=FileReadDatetime(file_handle);
         //--- signal value
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- set the indicator values that will not be seen on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      upbuff[i]=0;
      downbuff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, use the value from the file
            if(time[i]==time_buff[j])
              {
               //--- draw the arrow according to the signal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Type bool](), [FileWrite]()

FileReadDatetime

Reads from the file of CSV type a string of one of the formats: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" or "HH:MI:SS" - and converts it into a value of datetime type.

```
datetime  FileReadDatetime(
   int   file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of datetime type.

Example (the file obtained after executing the example for FileWrite function is used here)

```
//+------------------------------------------------------------------+
//|                                       Demo_FileReadDateTime.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv";  // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;       // index
double   upbuff[];    // indicator buffers of up arrows
double   downbuff[];  // indicator buffer of down arrows
bool     sign_buff[]; // signal array (true - buy, false - sell)
```

```
datetime time_buff[]; // array of signals' arrival time
int      size=0;      // size of signal arrays
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- first, read the number of signals
      size=(int)FileReadNumber(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         //--- signal time
         time_buff[i]=FileReadDatetime(file_handle);
         //--- signal value
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- set the indicator values that will not be seen on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      upbuff[i]=0;
      downbuff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, use the value from the file
            if(time[i]==time_buff[j])
              {
               //--- draw the arrow according to the signal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Type datetime](), [StringToTime](), [TimeToString](), [FileWrite]()

FileReadDouble

Reads a double-precision floating point number (double) from the current position of the binary file.

```
double  FileReadDouble(
   int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of double type.

Note

For more details about the error, call GetLastError().

Example (the file obtained after executing the example for FileWriteDouble function is used here)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- data reading parameters
input string InpFileName="MA.csv";    // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;
int      size=0;
double   ma_buff[];
datetime time_buff[];
//--- indicator buffer
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- first, read the amount of data in the file
      size=(int)FileReadDouble(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(ma_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadDouble(file_handle);
         ma_buff[i]=FileReadDouble(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer with index 0
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will not be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      //--- check if any data still exists
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if the dates coincide, the value from the file is used
            if(time[i]==time_buff[j])
              {
               buff[i]=ma_buff[j];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Real types (double, float)](), [StringToDouble](), [DoubleToString](), [FileWriteDouble]()

FileReadFloat

Reads the single-precision floating point number (float) from the current position of the binary file.

```
float  FileReadFloat(
   int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of float type.

Note

For more details about the error, call GetLastError().

Example (the file obtained after executing the example for FileWriteFloat function is used here)

```
//+------------------------------------------------------------------+
//|                                            Demo_FileReadFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "CloseLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="Close.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;
int      size=0;
double   close_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
double   color_buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- allocate memory for the arrays
   ArrayResize(close_buff,def_size);
   ArrayResize(time_buff,def_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- read time and price values
         time_buff[size]=(datetime)FileReadDouble(file_handle);
         close_buff[size]=(double)FileReadFloat(file_handle);
         size++;
         //--- increase the array sizes if they are overflown
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(close_buff,def_size);
            ArrayResize(time_buff,def_size);
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- bind the arrays to the indicator buffers
   SetIndexBuffer(0,buff,INDICATOR_DATA);
   SetIndexBuffer(1,color_buff,INDICATOR_COLOR_INDEX);
//---- set the indicator values that will not be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      color_buff[i]=0; // red color by default
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if the dates coincide, the value from the file is used
            if(time[i]==time_buff[j])
              {
               //--- receive the price
               buff[i]=close_buff[j];
               //--- if the current price exceeds the previous one, the color is blue
               if(buff[i-1]>buff[i])
                  color_buff[i]=1;
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Real types (double, float), FileReadDouble, FileWriteFloat

FileReadInteger

The function reads int, short or char value from the current position of the file pointer depending on the length specified in bytes.

```
int  FileReadInteger(
   int  file_handle,         // File handle
   int  size=INT_VALUE       // Size of an integer in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive) that should be read. The corresponding constants are provided: CHAR_VALUE = 1, SHORT_VALUE = 2 and INT_VALUE = 4, so the function can read the whole value of char, short or int type.

Return Value

A value of the int type. The result of this function must be explicitly cast to a target type, i.e. to the type of data that you need to read. Since a value of the int type is returned, it can be easily converted to any integer value. The file pointer is shifted by the number of bytes read.

Note

When reading less than 4 bytes, the received result is always positive. If one or two bytes are read, the sign of the number can be determined by explicit casting to type char (1 byte) or short (2 bytes). Getting the sign for a three-byte number is not trivial, since there is no corresponding underlying type.

Example (the file obtained after executing the example for FileWriteInteger function is used here)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```
//--- parameters for data reading
input string InpFileName="Trend.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;
int      size=0;
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- allocate memory for the array
   ArrayResize(time_buff,def_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- additional variables
      int    arr_size;
      uchar  arr[];
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- find out how many symbols are used for writing the time
         arr_size=FileReadInteger(file_handle,INT_VALUE);
         ArrayResize(arr,arr_size);
         for(int i=0;i<arr_size;i++)
            arr[i]=(char)FileReadInteger(file_handle,CHAR_VALUE);
         //--- store the time value
         time_buff[size]=StringToTime(CharArrayToString(arr));
         size++;
         //--- increase the sizes of the arrays if they are overflown
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(time_buff,def_size);
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will not be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(close,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, the value from the file is used
            if(time[i]==time_buff[j])
              {
               //--- receive the price
               buff[i]=close[i];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

IntegerToString, StringToInteger, Integer types, FileWriteInteger

FileReadLong

The function reads an integer of long type (8 bytes) from the current position of the binary file.

```
long  FileReadLong(
   int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of long type.

Example (the file obtained during the execution of an example for FileWriteLong function is used here)

```
//+------------------------------------------------------------------+
//|                                           Demo_FileReadLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Volume"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrYellow
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- parameters for data reading
input string InpFileName="Volume.bin"; // file name
input string InpDirectoryName="Data";  // directory name
//--- global variables
int      ind=0;
int      size=0;
long     volume_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
```

```
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- first, read the amount of data in the file
      size=(int)FileReadLong(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(volume_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadLong(file_handle);
         volume_buff[i]=FileReadLong(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer with 0 index
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, the value from the file is used
            if(time[i]==time_buff[j])
              {
               buff[i]=(double)volume_buff[j];
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Integer types, FileReadInteger, FileWriteLong

FileReadNumber

The function reads from the CSV file a string from the current position till a separator (or till the end of a text string) and converts the read string to a value of double type.

```
double  FileReadNumber(
   int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of double type.

Example (the file obtained during the execution of an example for FileWriteString function is used here)

```
//+------------------------------------------------------------------+
//|                                         Demo_FileReadNumber.mq5 |
//|                         Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="RSI.csv";   // file name
input string InpDirectoryName="Data"; // directory name
//--- indicator buffers
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- overbought variables
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
```

```
//--- oversold variables
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- variables of array sizes by default
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- allocate memory for arrays
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      double value;
      //--- read data from file
      while(!FileIsEnding(file_handle))
        {
         //--- read the first value in the string
         value=FileReadNumber(file_handle);
         //--- read to different arrays according to the function result
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
```

```
//---- set the indicator values that will not be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Read the file's string data                                      |
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- read till the end of the string or of the file is reached
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- shift the carriage after reading the number
      if(flag)
         FileReadNumber(file_handle);
      //--- store the current date
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- increase the array size if necessary
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- slip past the first iteration
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
```

```
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- check if any date is still present
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- if the dates coincide, the bar is in the overbought area
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - red color
               color_buff[i]=0;
               //--- increase the counter
               ovb_ind=j+1;
               break;
              }
           }
      //--- check if any data still exists
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- if the dates coincide, the bar is in the oversold area
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - blue color
               color_buff[i]=1;
               //--- increase the counter
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| ChartEvent event handler                                         |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
                  )
  {
//--- change the indicator width according to the scale
   if(ChartGetInteger(0,CHART_SCALE)>3)
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
   else
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
  }
```

See also

FileWriteString

FileReadString

The function reads a string from the current position of a file pointer in a file.

```
string  FileReadString(
   int  file_handle,      // File handle
   int  length=-1         // Length of the string
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

length=-1

[in] Number of characters to read.

Return Value

Line read (string).

Note

When reading from a bin-file. the length of a string to read must be specified. When reading from a txt-file the string length is not required, and the string will be read from the current position to the line feed character "\r\n". When reading from a csv-file, the string length isn't required also, the string will be read from the current position till the nearest delimiter or till the text string end character.

If the file is opened with FILE_ANSI flag, then the line read is converted to Unicode.

Example (the file obtained after executing the example for FileWriteInteger function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for data reading
input string InpFileName="Trend.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- additional variables
      int    str_size;
      string str;
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- find out how many symbols are used for writing the time
         str_size=FileReadInteger(file_handle,INT_VALUE);
         //--- read the string
         str=FileReadString(file_handle,str_size);
         //--- print the string
         PrintFormat(str);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

See also

String Type, Conversion Functions, FileWriteInteger

FileReadStruct

The function reads contents into a structure passed as a parameter from a binary-file, starting with the current position of the file pointer.

```
uint  FileReadStruct(
   int          file_handle,        // file handle
   const void&  struct_object,      // target structure to which the contents are reac
   int          size=-1             // structure size in bytes
   );
```

Parameters

file_handle

[in] File descriptor of an open bin-file.

struct_object

[out] The object of this structure. The structure should not contain strings, dynamic arrays or virtual functions.

size=-1

[in] Number of bytes that should be read. If size is not specified or the indicated value is greater than the size of the structure, the exact size of the specified structure is used.

Return Value

If successful the function returns the number of bytes read. File pointer is moved by the same number of bytes.

Example (the file obtained after using the example for FileWriteStruct function is used here)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Candles"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
//--- parameters for receiving data
input string  InpFileName="EURUSD.txt"; // file name
input string  InpDirectoryName="Data";  // directory name
```

```
//+-------------------------------------------------------------------+
//| Structure for storing candlestick data                           |
//+-------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // open price
   double            close; // close price
   double            high;  // high price
   double            low;   // low price
   datetime          date;  // date
  };
//--- indicator buffers
double       open_buff[];
double       close_buff[];
double       high_buff[];
double       low_buff[];
//--- global variables
candlesticks cand_buff[];
int          size=0;
int          ind=0;
//+-------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+-------------------------------------------------------------------+
int OnInit()
  {
   int default_size=100;
   ArrayResize(cand_buff,default_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- write data to the array
         FileReadStruct(file_handle,cand_buff[size]);
         size++;
         //--- check if the array is overflown
         if(size==default_size)
           {
            //--- increase the array size
            default_size+=100;
            ArrayResize(cand_buff,default_size);
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
      return(INIT_FAILED);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
//--- empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the candlesticks that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      open_buff[i]=0;
      close_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, the value from the file is used
            if(time[i]==cand_buff[j].date)
              {
                 open_buff[i]=cand_buff[j].open;
                 close_buff[i]=cand_buff[j].close;
                 high_buff[i]=cand_buff[j].high;
                 low_buff[i]=cand_buff[j].low;
                 //--- increase the counter
                 ind=j+1;
                 break;
                }
             }
          }
       }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

[Structures and classes](), [FileWriteStruct]()

FileSeek

The function moves the position of the file pointer by a specified number of bytes relative to the specified position.

```
bool  FileSeek(
   int                  file_handle,     // File handle
   long                 offset,          // In bytes
   ENUM_FILE_POSITION   origin           // Position for reference
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

offset

[in] The shift in bytes (may take a negative value).

origin

[in] The starting point for the displacement. Can be one of values of ENUM_FILE_POSITION.

Return Value

If successful the function returns true, otherwise false. To obtain information about the error call the GetLastError() function.

Note

If the execution of the FileSeek() function results in a negative shift (going beyond the "level boundary" of the file), the file pointer will be set to the file beginning.

If a position is set beyond the "right boundary" of the file (larger than the file size), the next writing to the file will be performed not from the end of the file, but from the position set. In this case indefinite values will be written for the previous file end and the position set.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_FileSeek.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- specify the value of the variable for generating random numbers
   _RandomSeed=GetTickCount();
//--- variables for positions of the strings' start points
   ulong pos[];
   int   size;
//--- reset the error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- receive start position for each string in the file
      GetStringPositions(file_handle,pos);
      //--- define the number of strings in the file
      size=ArraySize(pos);
      if(!size)
        {
         //--- stop if the file does not have strings
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- make a random selection of a string number
      int ind=MathRand()%size;
      //--- shift position to the starting point of the string
      if(FileSeek(file_handle,pos[ind],SEEK_SET)==true)
        {
      //--- read and print the string with ind number
         PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_han
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
//+------------------------------------------------------------------------------+
//| The function defines starting points for each of the strings in the file and  |
//| places them in arr array                                                     |
//+------------------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
//--- default array size
```

```
   int def_size=127;
//--- allocate memory for the array
   ArrayResize(arr,def_size);
//--- string counter
   int i=0;
//--- if this is not the file's end, then there is at least one string
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // the file is empty, exit
//--- define the shift in bytes depending on encoding
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- go through the strings in the loop
   while(1)
     {
      //--- read the string
      FileReadString(handle);
      //--- check for the file end
      if(!FileIsEnding(handle))
        {
         //--- store the next string's position
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- increase the size of the array if it is overflown
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // end of the file, exit
     }
//--- define the actual size of the array
   ArrayResize(arr,i);
  }
```

FileSize

The function returns the file size in bytes.

```
ulong  FileSize(
   int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of type int.

Note

To obtain information about the error call GetLastError().

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input ulong  InpThresholdSize=20;        // file threshold size in kilobytes
input string InpBigFolderName="big";     // folder for large files
input string InpSmallFolderName="small"; // folder for small files
//+-------------------------------------------------------------------+
//| Script program start function                                     |
//+-------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.csv"; // filter for searching the files
   ulong    file_size=0;    // file size in bytes
   int      size=0;         // number of files
//--- print the path to the file we are going to work with
   PrintFormat("Working in %s\\Files\\ folder",TerminalInfoString(TERMINAL_COMMONDATA_
//--- receive the search handle in common folder's root of all terminals
   long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);
//--- check if FileFindFirst() has been executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- move files in the loop according to their size
      do
        {
         //--- open the file
         ResetLastError();
         int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON);
         if(file_handle!=INVALID_HANDLE)
```

```
           {
            //--- receive the file size
            file_size=FileSize(file_handle);
            //--- close the file
            FileClose(file_handle);
           }
         else
           {
            PrintFormat("Failed to open %s file, Error code = %d",file_name,GetLastEr
            continue;
           }
         //--- print the file size
         PrintFormat("Size of %s file is equal to %d bytes",file_name,file_size);
         //--- define the path for moving the file
         string path;
         if(file_size>InpThresholdSize*1024)
            path=InpBigFolderName+"//"+file_name;
         else
            path=InpSmallFolderName+"//"+file_name;
         //--- move the file
         ResetLastError();
         if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))
            PrintFormat("%s file is moved",file_name);
         else
            PrintFormat("Error, code = %d",GetLastError());
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

FileTell

The file returns the current position of the file pointer of an open file.

```
ulong  FileTell(
   int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Current position of the file descriptor in bytes from the beginning of the file.

Note

To obtain information about the error call GetLastError().

Example:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileTell.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- specify the value of the variable for generating random numbers
   _RandomSeed=GetTickCount();
//--- variables for positions of the strings' start points
   ulong pos[];
   int   size;
//--- reset the error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
   if(file_handle!=INVALID_HANDLE)
```

```
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- receive start position for each string in the file
      GetStringPositions(file_handle,pos);
      //--- define the number of strings in the file
      size=ArraySize(pos);
      if(!size)
        {
         //--- stop if the file does not have strings
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- make a random selection of a string number
      int ind=MathRand()%size;
      //--- shift position to the starting point of the string
      FileSeek(file_handle,pos[ind],SEEK_SET);
      //--- read and print the string with ind number
      PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(file_handle)
      //--- close the file
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
//+------------------------------------------------------------------+
//| The function defines starting points for each of the strings in the file and  |
//| places them in arr array                                          |
//+------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
//--- default array size
   int def_size=127;
//--- allocate memory for the array
   ArrayResize(arr,def_size);
//--- string counter
   int i=0;
//--- if this is not the file's end, then there is at least one string
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // the file is empty, exit
//--- define the shift in bytes depending on encoding
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
```

```
        shift=1;
     else
        shift=2;
//--- go through the strings in the loop
   while(1)
     {
      //--- read the string
      FileReadString(handle);
      //--- check for the file end
      if(!FileIsEnding(handle))
        {
         //--- store the next string's position
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- increase the size of the array if it is overflown
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // end of the file, exit
     }
//--- define the actual size of the array
   ArrayResize(arr,i);
  }
```

FileWrite

The function is intended for writing of data into a CSV file, delimiter being inserted automatically unless it is equal to 0. After writing into the file, the line end character "\r\n" will be added.

```
uint  FileWrite(
   int  file_handle,    // File handle
   ...                  // List of recorded parameters
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

...

[in] The list of parameters separated by commas, to write to the file. The number of written parameters can be up to 63.

Return Value

Number of bytes written.

Note

Numbers will be converted into a text at output (see the Print() function). Data of the double type are output with the accuracy of 16 digits after the decimal point, and the data can be displayed either in traditional or in scientific format - depending on which format will be the most compact. The data of the float type are shown with 5 digits after the decimal point. To output real numbers with different precision or in a clearly specified format, use DoubleToString().

Numbers of the bool type are displayed as "true" or "false" strings. Numbers of the datetime type are displayed as "YYYY.MM.DD HH:MI:SS".

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_FileWrite.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // time frame
input int                InpFastEMAPeriod=12;              // fast EMA period
input int                InpSlowEMAPeriod=26;              // slow EMA period
input int                InpSignalPeriod=9;                // difference averaging pe
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // price type
input datetime           InpDateStart=D'2012.01.01 00:00'; // data copying start date
//--- parameters for writing data to file
input string             InpFileName="MACD.csv";  // file name
input string             InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // data copying end date
   bool     sign_buff[]; // signal array (true - buy, false - sell)
   datetime time_buff[]; // array of signals' arrival time
   int      sign_size=0; // signal array size
   double   macd_buff[]; // array of indicator values
   datetime date_buff[]; // array of indicator dates
   int      macd_size=0; // size of indicator arrays
//--- end time is the current time
   date_finish=TimeCurrent();
//--- receive MACD indicator handle
   ResetLastError();
   int macd_handle=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,InpSlowEMAPeri
   if(macd_handle==INVALID_HANDLE)
     {
      //--- failed to receive indicator handle
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErro
      return;
     }
//--- being in the loop until the indicator calculates all its values
   while(BarsCalculated(macd_handle)==-1)
      Sleep(10); // pause to allow the indicator to calculate all its values
//--- copy the indicator values for a certain period of time
   ResetLastError();
   if(CopyBuffer(macd_handle,0,InpDateStart,date_finish,macd_buff)==-1)
     {
      PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
      return;
     }
//--- copy the appropriate time for the indicator values
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- free the memory occupied by the indicator
   IndicatorRelease(macd_handle);
//--- receive the buffer size
   macd_size=ArraySize(macd_buff);
//--- analyze the data and save the indicator signals to the arrays
```

```
   ArrayResize(sign_buff,macd_size-1);
   ArrayResize(time_buff,macd_size-1);
   for(int i=1;i<macd_size;i++)
     {
      //--- buy signal
      if(macd_buff[i-1]<0 && macd_buff[i]>=0)
        {
         sign_buff[sign_size]=true;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
      //--- sell signal
      if(macd_buff[i-1]>0 && macd_buff[i]<=0)
        {
         sign_buff[sign_size]=false;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
     }
//--- open the file for writing the indicator values (if the file is absent, it will
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- first, write the number of signals
      FileWrite(file_handle,sign_size);
      //--- write the time and values of signals to the file
      for(int i=0;i<sign_size;i++)
         FileWrite(file_handle,time_buff[i],sign_buff[i]);
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError(
  }
```

See also

Comment, Print, StringFormat

FileWriteArray

The function writes arrays of any type except for string to a BIN file (can be an array of structures not containing strings or dynamic arrays).

```
uint  FileWriteArray(
   int           file_handle,          // File handle
   const void&   array[],              // Array
   int           start=0,              // Start index in the array
   int           count=WHOLE_ARRAY     // Number of elements
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

array[]

[out] Array for recording.

start=0

[in] Initial index in the array (number of the first recorded element).

count=WHOLE_ARRAY

[in] Number of items to write (WHOLE_ARRAY means that all items starting with the number start until the end of the array will be written).

Return Value

Number of recorded items.

Note

String array can be recorded in a TXT file. In this case, strings are automatically ended by the line end characters "\r\n". Depending on the file type ANSI or UNICODE, strings are either converted to ansi-encoding or not.

Example:

```
//+------------------------------------------------------------------+
//|                                           Demo_FileWriteArray.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Structure for storing price data                                 |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // date
   double            bid;  // bid price
   double            ask;  // ask price
  };
//--- global variables
int    count=0;
int    size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- allocate memory for the array
   ArrayResize(arr,size);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- write the remaining count strings if count<n
   WriteData(count);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- save data to array
   arr[count].date=TimeCurrent();
   arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- show current data
   Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[count].ask);
//--- increase the counter
   count++;
//--- if the array is filled, write data to the file and zero it out
   if(count==size)
     {
      WriteData(size);
      count=0;
     }
  }
```

```
//+------------------------------------------------------------------+
//| Write n elements of the array to file                            |
//+------------------------------------------------------------------+
void WriteData(const int n)
  {
//--- open the file
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
   if(handle!=INVALID_HANDLE)
     {
      //--- write array data to the end of the file
      FileSeek(handle,0,SEEK_END);
      FileWriteArray(handle,arr,0,n);
      //--- close the file
      FileClose(handle);
     }
   else
      Print("Failed to open the file, error ",GetLastError());
  }
```

See also

Variables, FileSeek

FileWriteDouble

The function writes the value of a double parameter to a a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteDouble(
   int      file_handle,      // File handle
   double   value             // Value to write
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in]   The value of double type.

Return Value

If successful the function returns the number of bytes written (in this case sizeof(double)=8). The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWriteDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURJPY";           // currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_M15;       // time frame
input int                InpMAPeriod=10;                   // smoothing period
input int                InpMAShift=0;                     // indicator shift
input ENUM_MA_METHOD     InpMAMethod=MODE_SMA;             // smoothing type
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // price type
input datetime           InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string             InpFileName="MA.csv";    // file name
input string             InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   ma_buff[];
   datetime time_buff[];
   int      size;
//--- receive MA indicator handle
   ResetLastError();
   int ma_handle=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,InpMAMethod,
   if(ma_handle==INVALID_HANDLE)
     {
      //--- failed to receive the indicator handle
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErr
      return;
     }
//--- being in the loop until the indicator calculates all its values
   while(BarsCalculated(ma_handle)==-1)
      Sleep(20); // a pause to allow the indicator to calculate all its values
   PrintFormat("Indicator values starting from %s will be written to the file",TimeTo
//--- copy the indicator values
   ResetLastError();
   if(CopyBuffer(ma_handle,0,InpDateStart,date_finish,ma_buff)==-1)
     {
      PrintFormat("Failed to copy the indicator values. Error code = %d",GetLastError
      return;
     }
//--- copy the time of the appropriate bars' arrival
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- receive the buffer size
   size=ArraySize(ma_buff);
//--- free the memory occupied by the indicator
   IndicatorRelease(ma_handle);
//--- open the file for writing the indicator values (if the file is absent, it will
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- first, write the size of data sample
      FileWriteDouble(file_handle,(double)size);
      //--- write the indicator time and value to the file
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteDouble(file_handle,ma_buff[i]);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

See also

[Real types (double, float)]

FileWriteFloat

The function writes the value of the float parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteFloat(
   int     file_handle,     // File handle
   float   value            // Value to be written
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] The value of float type.

Return Value

If successful the function returns the number of bytes written (in this case sizeof(float)=4). The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;       // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string          InpFileName="Close.bin"; // file name
input string          InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
```

```
//--- reset the error value
   ResetLastError();
//--- copy the close price for each bar
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Failed to copy close price values. Error code = %d",GetLastError()
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy the time values. Error code = %d",GetLastError());
      return;
     }
//--- receive the buffer size
   size=ArraySize(close_buff);
//--- open the file for writing the values (if the file is absent, it will be created
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- write close prices' time and values to the file
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteFloat(file_handle,(float)close_buff[i]);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

See also

Real types (double, float), FileWriteDouble

FileWriteInteger

The function writes the value of the int parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteInteger(
   int  file_handle,        // File handle
   int  value,              // Value to be written
   int  size=INT_VALUE      // Size in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] Integer value.

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive), that should be written. The corresponding constants are provided: CHAR_VALUE=1, SHORT_VALUE=2 and INT_VALUE=4, so the function can write the integer value of char, uchar, short, ushort, int, or uint type.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                       Demo_FileWriteInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // time frame
input datetime           InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string             InpFileName="Trend.bin"; // file name
input string             InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- reset the error value
   ResetLastError();
//--- copy the close price for each bar
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Failed to copy the values of close prices. Error code = %d",GetLast
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- receive the buffer size
   size=ArraySize(close_buff);
//--- open the file for writing the values (if the file is absent, it will be created
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //---
      int   up_down=0; // trend flag
      int   arr_size;  // arr array size
      uchar arr[];     // uchar type array
      //--- write time values to the file
      for(int i=0;i<size-1;i++)
        {
         //--- compare close prices of the current and next bars
         if(close_buff[i]<=close_buff[i+1])
           {
            if(up_down!=1)
              {
               //--- write date value to the file using FileWriteInteger
               StringToCharArray(TimeToString(time_buff[i]),arr);
               arr_size=ArraySize(arr);
               //--- first, write the number of symbols in the array
               FileWriteInteger(file_handle,arr_size,INT_VALUE);
               //--- write the symbols
               for(int j=0;j<arr_size;j++)
                  FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
               //--- change the trend flag
```

```
             up_down=1;
            }
         }
       else
         {
          if(up_down!=-1)
            {
             //--- write the date value to the file using FileWriteInteger
             StringToCharArray(TimeToString(time_buff[i]),arr);
             arr_size=ArraySize(arr);
             //--- first, write the number of symbols in the array
             FileWriteInteger(file_handle,arr_size,INT_VALUE);
             //--- write the symbols
             for(int j=0;j<arr_size;j++)
                FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
             //--- change the trend flag
             up_down=-1;
            }
         }
      }
    //--- close the file
    FileClose(file_handle);
    PrintFormat("Data is written, %s file is closed",InpFileName);
   }
 else
    PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
}
```

See also

[IntegerToString](), [StringToInteger](), [Integer types]()

FileWriteLong

The function writes the value of the long-type parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteLong(
   int    file_handle,      // File handle
   long   value             // Value to be written
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] Value of type long.

Return Value

If successful the function returns the number of bytes written (in this case sizeof(long)=8). The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string          InpFileName="Volume.bin"; // file name
input string          InpDirectoryName="Data";  // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   long     volume_buff[];
   datetime time_buff[];
   int      size;
```

```
//--- reset the error value
   ResetLastError();
//--- copy tick volumes for each bar
   if(CopyTickVolume(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,volume_bu
     {
      PrintFormat("Failed to copy values of the tick volume. Error code = %d",GetLast
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- receive the buffer size
   size=ArraySize(volume_buff);
//--- open the file for writing the indicator values (if the file is absent, it will 
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- first, write the data sample size
      FileWriteLong(file_handle,(long)size);
      //--- write time and volume values to file
      for(int i=0;i<size;i++)
        {
         FileWriteLong(file_handle,(long)time_buff[i]);
         FileWriteLong(file_handle,volume_buff[i]);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError(
  }
```

See also

[Integer types](), [FileWriteInteger]()

FileWriteString

The function writes the value of a string-type parameter into a BIN, CSV or TXT file starting from the current position of the file pointer. When writing to a CSV or TXT file: if there is a symbol in the string '\n' (LF) without previous character '\r' (CR), then before '\n' the missing '\r' is added.

```
uint  FileWriteString(
   int            file_handle,     // File handle
   const string   text_string,     // string to write
   int            length=-1        // number of symbols
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

text_string

[in] String.

length=-1

[in] The number of characters that you want to write. This option is needed for writing a string into a BIN file. If the size is not specified, then the entire string without the trailer 0 is written. If you specify a size smaller than the length of the string, then a part of the string without the trailer 0 is written. If you specify a size greater than the length of the string, the string is filled by the appropriate number of zeros. For files of CSV and TXT type, this parameter is ignored and the string is written entirely.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Note

Note that when writing to a file opened by the FILE_UNICODE flag (or without a flag FILE_ANSI), then the number of bytes written will be twice as large as the number of string characters written. When recording to a file opened with the FILE_ANSI flag, the number of bytes written will coincide with the number of string characters written.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteString.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
```

```
input string               InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES      InpSymbolPeriod=PERIOD_H1;        // time frame
input int                  InpMAPeriod=14;                   // MA period
input ENUM_APPLIED_PRICE   InpAppliedPrice=PRICE_CLOSE;      // price type
input datetime             InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string               InpFileName="RSI.csv";   // file name
input string               InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // data copying end date
   double   rsi_buff[];  // array of indicator values
   datetime date_buff[]; // array of the indicator dates
   int      rsi_size=0;  // size of the indicator arrays
//--- end time is the current one
   date_finish=TimeCurrent();
//--- receive RSI indicator handle
   ResetLastError();
   int rsi_handle=iRSI(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- failed to receive the indicator handle
      PrintFormat("Error when receiving indicator handle. Error code = %d",GetLastErr
      return;
     }
//--- being in the loop, until the indicator calculates all its values
   while(BarsCalculated(rsi_handle)==-1)
      Sleep(10); // a pause to allow the indicator to calculate all its values
//--- copy the indicator values for a certain period of time
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,date_finish,rsi_buff)==-1)
     {
      PrintFormat("Failed to copy indicator values. Error code = %d",GetLastError());
      return;
     }
//--- copy the appropriate time for the indicator values
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- free the memory occupied by the indicator
   IndicatorRelease(rsi_handle);
//--- receive the buffer size
   rsi_size=ArraySize(rsi_buff);
```

```
//--- open the file for writing the indicator values (if the file is absent, it will
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
      //--- prepare additional variables
      string str="";
      bool   is_formed=false;
      //--- write dates of forming overbought and oversold areas
      for(int i=0;i<rsi_size;i++)
        {
         //--- check the indicator values
         if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
           {
            //--- if the value is the first one in this area
            if(!is_formed)
              {
               //--- add the value and the date
               str=(string)rsi_buff[i]+"\t"+(string)date_buff[i];
               is_formed=true;
              }
            else
               str+="\t"+(string)rsi_buff[i]+"\t"+(string)date_buff[i];
            //--- move to the next loop iteration
            continue;
           }
         //--- check the flag
         if(is_formed)
           {
            //--- the string is formed, write it to the file
            FileWriteString(file_handle,str+"\r\n");
            is_formed=false;
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

See also

String Type, StringFormat

FileWriteStruct

The function writes into a bin-file contents of a structure passed as a parameter, starting from the current position of the file pointer.

```
uint  FileWriteStruct(
   int          file_handle,       // File handle
   const void&  struct_object,     // link to an object
   int          size=-1            // size to be written in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

struct_object

[in] Reference to the object of this structure. The structure should not contain strings, dynamic arrays or virtual functions.

size=-1

[in] Number of bytes that you want to record. If size is not specified or the specified number of bytes is greater than the size of the structure, the entire structure is written.

Return Value

If successful the function returns the number of bytes written. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWiteStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying start date
//--- parameters for writing data to the file
input string          InpFileName="EURUSD.txt";         // file name
input string          InpDirectoryName="Data";          // directory name
//+------------------------------------------------------------------+
//| Structure for storing candlestick data                           |
//+------------------------------------------------------------------+
struct candlesticks
```

```
  {
   double               open;   // open price
   double               close;  // close price
   double               high;   // high price
   double               low;    // low price
   datetime             date;   // date
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime     date_finish=TimeCurrent();
   int          size;
   datetime     time_buff[];
   double       open_buff[];
   double       close_buff[];
   double       high_buff[];
   double       low_buff[];
   candlesticks cand_buff[];
//--- reset the error value
   ResetLastError();
//--- receive the time of the arrival of the bars from the range
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastError());
      return;
     }
//--- receive high prices of the bars from the range
   if(CopyHigh(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,high_buff)==-1)
     {
      PrintFormat("Failed to copy values of high prices. Error code = %d",GetLastErro
      return;
     }
//--- receive low prices of the bars from the range
   if(CopyLow(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,low_buff)==-1)
     {
      PrintFormat("Failed to copy values of low prices. Error code = %d",GetLastError
      return;
     }
//--- receive open prices of the bars from the range
   if(CopyOpen(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,open_buff)==-1)
     {
      PrintFormat("Failed to copy values of open prices. Error code = %d",GetLastErro
      return;
     }
//--- receive close prices of the bars from the range
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Failed to copy values of close prices. Error code = %d",GetLastErro
      return;
     }
//--- define dimension of the arrays
   size=ArraySize(time_buff);
//--- save all data in the structure array
   ArrayResize(cand_buff,size);
   for(int i=0;i<size;i++)
     {
      cand_buff[i].open=open_buff[i];
      cand_buff[i].close=close_buff[i];
      cand_buff[i].high=high_buff[i];
      cand_buff[i].low=low_buff[i];
      cand_buff[i].date=time_buff[i];
     }

//--- open the file for writing the structure array to the file (if the file is absent
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH
      //--- prepare the counter of the number of bytes
      uint counter=0;
      //--- write array values in the loop
      for(int i=0;i<size;i++)
         counter+=FileWriteStruct(file_handle,cand_buff[i]);
      PrintFormat("%d bytes of information is written to %s file",InpFileName,counter)
      PrintFormat("Total number of bytes: %d * %d * %d = %d, %s",size,5,8,size*5*8,si
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,GetLastError()
  }
```

See also

[Structures and classes]()

FileLoad

Reads all data of a specified binary file into a passed array of numeric types or simple structures. The function allows you to quickly read data of a known type into the appropriate array.

```
long  FileLoad(
   const string  file_name,         // File name
   void&         buffer[],          // An array of numeric types or simple structures
   int           common_flag=0      // A file flag, is searched in <data_folder>\MQL5\
   );
```

Parameters

file_name

[in] The name of the file from which data will be read.

buffer

[out] An array of numeric types or simple structures.

common_flag=0

[in] A file flag indicating the operation mode. If the parameter is not specified, the file is searched in the subfolder MQL5\Files (or in <testing_agent_directory>\MQL5\Files in case of testing).

Return Value

The number of elements read or -1 in case of an error.

Note

The FileLoad() function reads from a file the number of bytes multiple of the array element size. Suppose the file size is 10 bytes, and the function reads data into an array of type double (sizeof(double)=8). In this case the function will read only 8 bytes, the remaining 2 bytes at the end of the file will be dropped, and the function FileLoad() will return 1 (1 element read).

Example:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileLoad.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int      bars_to_save=10; // Number of bars
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string  filename=_Symbol+"_rates.bin";
   MqlRates rates[];
//---
   int copied=CopyRates(_Symbol,_Period,0,bars_to_save,rates);
   if(copied!=-1)
     {
      PrintFormat(" CopyRates(%s) copied %d bars",_Symbol,copied);
      //---  Writing quotes to a file
      if(!FileSave(filename,rates,FILE_COMMON))
         PrintFormat("FileSave() failed, error=%d",GetLastError());
     }
   else
      PrintFormat("Failed CopyRates(%s), error=",_Symbol,GetLastError());
//--- Now reading these quotes back to the file
   ArrayFree(rates);
   long count=FileLoad(filename,rates,FILE_COMMON);
   if(count!=-1)
     {
      Print("Time\tOpen\tHigh\tLow\tClose\tTick Voulme\tSpread\tReal Volume");
      for(int i=0;i<count;i++)
        {
         PrintFormat("%s\t%G\t%G\t%G\t%G\t%I64u\t%d\t%I64u",
                     TimeToString(rates[i].time,TIME_DATE|TIME_SECONDS),
                     rates[i].open,rates[i].high,rates[i].low,rates[i].close,
                     rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
     }
  }
```

See also

Structures and Classes, FileReadArray, FileReadStruct, FileSave

FileSave

Writes to a binary file all elements of an array passed as a parameter. The function allows you to quickly write arrays of numeric types or simple structures as one string.

```
bool  FileSave(
   const string  file_name,          // File name
   void&         buffer[],           // An array of numeric types or simple structures
   int           common_flag=0       // A file flag, by default files are written to <
   );
```

Parameters

file_name

[in] The name of the file, to the data array will be written.

buffer

[in] An array of numeric types or simple structures.

common_flag=0

[in] A file flag indicating the operation mode. If the parameter is not specified, the file will be written to the subfolder MQL5\Files (or to <testing_agent_directory>\MQL5\Files in case of testing).

Return Value

In case of failure returns false.

Example:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileSave.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int      ticks_to_save=1000; // Number of ticks
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string  filename=_Symbol+"_ticks.bin";
   MqlTick ticks[];
u//---
   int copied=CopyTicks(_Symbol,ticks,COPY_TICKS_ALL,0,ticks_to_save);
   if(copied!=-1)
     {
      PrintFormat(" CopyTicks(%s) copied %d ticks",_Symbol,copied);
      //--- If the tick history is synchronized, the error code is equal to zero
      if(!GetLastError()==0)
         PrintFormat("%s: Ticks are not synchronized, error=%d",_Symbol,copied,_LastE
      //---  Writing ticks to a file
      if(!FileSave(filename,ticks,FILE_COMMON))
         PrintFormat("FileSave() failed, error=%d",GetLastError());
     }
   else
      PrintFormat("Failed CopyTicks(%s), Error=",_Symbol,GetLastError());
//--- Now reading the ticks back to the file
   ArrayFree(ticks);
   long count=FileLoad(filename,ticks,FILE_COMMON);
   if(count!=-1)
     {
      Print("Time\tBid\tAsk\tLast\tVolume\tms\tflags");
      for(int i=0;i<count;i++)
        {
         PrintFormat("%s.%03I64u:\t%G\t%G\t%G\t%I64u\t0x%04x",
         TimeToString(ticks[i].time,TIME_DATE|TIME_SECONDS),ticks[i].time_msc%1000,
         ticks[i].bid,ticks[i].ask,ticks[i].last,ticks[i].volume,ticks[i].flags);
        }
     }
  }
```

See also

Structures and Classes, FileWriteArray, FileWriteStruct, FileLoad, FileWrite

FolderCreate

Creates a directory in the Files folder (depending on the common_flag value)

```
bool  FolderCreate(
   string  folder_name,        // line with the created folder name
   int     common_flag=0       // action area
   );
```

Parameters

folder_name

[in] Name of the directory to be created. Contains the relative path to the folder.

common_flag=0

[in] Flag defining the directory location. If common_flag=FILE_COMMON, the directory is located in the common folder of all client terminals \Terminal\Common\Files. Otherwise, the directory is in the local folder (MQL5\Files or MQL5\Tester\Files when testing).

Return Value

Returns true if successful, otherwise false.

Note

For reasons of security, working with files is strictly controlled in MQL5 language. Files used in file operations by means of MQL5 language cannot be located outside the file sandbox.

Example:

```
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- description
#property description "The script shows FolderCreate() application sample."
#property description "The external parameter defines the directory for creating folde
#property description "The folder structure is created after executing the script"

//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- the input parameter defines the folder, in which the script works
input bool     common_folder=false; // common folder for all terminals
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- folder to be created in MQL5\Files
   string root_folder="Folder_A";
   if(CreateFolder(root_folder,common_folder))
     {
      //--- create the Child_Folder_B1 sub-folder in it
      string folder_B1="Child_Folder_B1";
      string path=root_folder+"\\"+folder_B1;          // create the folder name cons
      if(CreateFolder(path,common_folder))
        {
         //--- create 3 more sub-directories in this folder
         string folder_C11="Child_Folder_C11";
         string child_path=root_folder+"\\"+folder_C11;// create the folder name cons
         CreateFolder(child_path,common_folder);
         //--- second sub-directory
         string folder_C12="Child_Folder_C12";
         child_path=root_folder+"\\"+folder_C12;
         CreateFolder(child_path,common_folder);

         //--- third sub-directory
         string folder_C13="Child_Folder_C13";
         child_path=root_folder+"\\"+folder_C13;
         CreateFolder(child_path,common_folder);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| Try creating a folder and display a message about that           |
//+------------------------------------------------------------------+
bool CreateFolder(string folder_path,bool common_flag)
  {
   int flag=common_flag?FILE_COMMON:0;
   string working_folder;
//--- define the full path depending on the common_flag parameter
   if(common_flag)
      working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL5\\Files";
   else
      working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- debugging message
   PrintFormat("folder_path=%s",folder_path);
//--- attempt to create a folder relative to the MQL5\Files path
   if(FolderCreate(folder_path,flag))
     {
      //--- display the full path for the created folder
      PrintFormat("Created the folder %s",working_folder+"\\"+folder_path);
      //--- reset the error code
      ResetLastError();
      //--- successful execution
      return true;
     }
   else
      PrintFormat("Failed to create the folder %s. Error code %d",working_folder+folde
//--- execution failed
   return false;
  }
```

See also

FileOpen(), FolderClean(), FileCopy()

FolderDelete

The function removes the specified directory. If the folder is not empty, then it can't be removed.

```
bool  FolderDelete(
   string  folder_name,        // String with the name of the folder to delete
   int     common_flag=0       // Scope
   );
```

Parameters

folder_name

[in] The name of the directory you want to delete. Contains the full path to the folder.

common_flag=0

[in] Flag determining the location of the directory. If common_flag=FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder (MQL5\Files or MQL5\Tester\Files in the case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

If the directory contains at least one file and/or subdirectory, then this directory can't be deleted, it must be cleared first. FolderClean() is used to clear a folder of all its files or subfolders.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FolderDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderDelete()."
#property description "First two folders are created; one of them is empty, the secon
#property description "When trying to delete a non-empty folder, an error is returned

//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- Input parameters
input string   firstFolder="empty";     // An empty folder
input string   secondFolder="nonempty";// The folder, in which one file will be create
string filename="delete_me.txt";        // The name of the file that will be created i
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Write the file handle here
   int handle;
//--- Find out in what folder we are working
   string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- A debug message
   PrintFormat("working_folder=%s",working_folder);
//--- Trying to create an empty folder relative to path MQL5\Files
   if(FolderCreate(firstFolder,0)) // 0 means that we are working in the local folder
     {
      //--- Enter the full path to the created folder
      PrintFormat("Folder %s has been created",working_folder+"\\"+firstFolder);
      //--- Reset the error code
      ResetLastError();
     }
   else
      PrintFormat("Failed to create folder %s. Error code %d",working_folder+"\\"+firs

//--- Now create a non-empty folder using the FileOpen() function
   string filepath=secondFolder+"\\"+filename;  // Form path to file that we want to
   handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // Flag FILE_WRITE in this case is
   if(handle!=INVALID_HANDLE)
      PrintFormat("File %s has been opened for reading",working_folder+"\\"+filepath);
   else
      PrintFormat("Failed to create file %s in folder %s. Error code=",filename,secon

   Comment(StringFormat("Prepare to delete folders %s and %s", firstFolder, secondFol
//--- A small pause of 5 seconds to read a message in the chart
   Sleep(5000); // Sleep() cannot be used in indicators!

//--- Show a dialog and ask the user
   int choice=MessageBox(StringFormat("Do you want to delete folders %s and %s?", firs
                         "Deleting folders",
                         MB_YESNO|MB_ICONQUESTION); //  Two buttons - "Yes" and "No"

//--- Run an action depending on the selected variant
   if(choice==IDYES)
     {
      //--- Delete the comment form the chart
      Comment("");
      //--- Add a message into the "Experts" journal
      PrintFormat("Trying to delete folders %s and %s",firstFolder, secondFolder);
      ResetLastError();
      //--- Delete the empty folder
      if(FolderDelete(firstFolder))
         //--- The following message should appear since the folder is empty
         PrintFormat("Folder %s has been successfully deleted",firstFolder);
      else
         PrintFormat("Failed to delete folder %s. Error code=%d", firstFolder, GetLast

      ResetLastError();
      //--- Delete the folder that contains a file
      if(FolderDelete(secondFolder))
         PrintFormat("Folder %s has been successfully deleted", secondFolder);
      else
         //--- The following message should appear since the folder contains a file
         PrintFormat("Failed to delete folder %s. Error code=%d", secondFolder, GetLas
     }
   else
      Print("Deletion canceled");
//---
  }
```

See also

FileOpen(), FolderClean(), FileMove()

FolderClean

The function deletes all files in a specified folder.

```
bool  FolderClean(
   string  folder_name,        // String with the name of the deleted folder
   int     common_flag=0       // Scope
   );
```

Parameters

folder_name

[in] The name of the directory where you want to delete all files. Contains the full path to the folder.

common_flag=0

[in] Flag determining the location of the directory. If common_flag = FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder(MQL5\Files or MQL5\Tester\Files in case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL5 language. Files with which file operations are conducted using MQL5 means, cannot be outside the file sandbox.

This function should be used with caution, since all the files and all subdirectories are deleted irretrievably.

Example:

```
//+------------------------------------------------------------------+
//|                                            Demo_FolderClean.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderClean()."
#property description "First, files are created in the specified folder using the File
#property description "Then, before the files are deleted, a warning is shown using Me

//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- Input parameters
input string   foldername="demo_folder";  // Create a folder in MQL5/Files/
input int      files=5;                   // The number of files to create and delete
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string name="testfile";
//--- First open or create files in the terminal data folder
   for(int N=0;N<files;N++)
     {
      //--- The name of the file in the form of 'demo_folder\testfileN.txt'
      string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
      //--- Open a file with the flag for writing, in this case the 'demo_folder' will
      int handle=FileOpen(filemane,FILE_WRITE);
      //--- Find out if the FileOpen() function was successful
      if(handle==INVALID_HANDLE)
        {
         PrintFormat("Failed to create file %s. Error code",filemane,GetLastError());
         ResetLastError();
        }
      else
        {
         PrintFormat("File %s has been successfully opened",filemane);
         //--- The opened file is not needed any more, so close it
         FileClose(handle);
        }
     }

//--- Check the number of files in the folder
   int k=FilesInFolder(foldername+"\\*.*",0);
   PrintFormat("Totally the folder %s contains %d files",foldername,k);

//--- Show a dialog to ask the user
   int choice=MessageBox(StringFormat("You are going to delete %d files from folder %s
                         "Deleting files from the folder",
                         MB_YESNO|MB_ICONQUESTION); //  Two buttons - "Yes" and "No"
   ResetLastError();

//--- Run an action depending on the selected variant
   if(choice==IDYES)
     {
      //--- Start to delete files
      PrintFormat("Trying to delete all files from folder %s",foldername);
      if(FolderClean(foldername,0))
         PrintFormat("Files have been successfully deleted, %d files left in folder %s
                     foldername,
                     FilesInFolder(foldername+"\\*.*",0));
      else
         PrintFormat("Failed to delete files from folder %s. Error code %d",foldername
     }
   else
      PrintFormat("Deletion canceled");
//---
  }
//+------------------------------------------------------------------+
//| Returns the number of files in the specified folder              |
//+------------------------------------------------------------------+
int FilesInFolder(string path,int flag)
  {
   int count=0;
   long handle;
   string filename;
//---
   handle=FileFindFirst(path,filename,flag);
//--- If at least one file found, search for more files
   if(handle!=INVALID_HANDLE)
     {
      //--- Show the name of the file
      PrintFormat("File %s found",filename);
      //--- Increase the counter of found files/folders
      count++;
      //--- Start search in all files/folders
      while(FileFindNext(handle,filename))
        {
         PrintFormat("File %s found",filename);
         count++;
        }
      //--- Do not forget to close the search handle upon completion
      FileFindClose(handle);
     }
   else // Failed to get the handle
     {
      PrintFormat("Files search in folder %s failed",path);
     }
//--- Return the result
   return count;
  }
```

See also

FileFindFirst, FileFindNext, FileFindClose

Custom Indicators

This is the group functions used in the creation of custom indicators. These functions can't be used when writing Expert Advisors and Scripts.

Function	Action
SetIndexBuffer	Binds the specified indicator buffer with one-dimensional dynamic array of the double type
IndicatorSetDouble	Sets the value of an indicator property of the double type
IndicatorSetInteger	Sets the value of an indicator property of the int type
IndicatorSetString	Sets the value of an indicator property of the string type
PlotIndexSetDouble	Sets the value of an indicator line property of the type double
PlotIndexSetInteger	Sets the value of an indicator line property of the int type
PlotIndexSetString	Sets the value of an indicator line property of the string type
PlotIndexGetInteger	Returns the value of an indicator line property of the integer type

Indicator properties can be set using the compiler directives or using functions. To better understand this, it is recommended that you study indicator styles in examples.

All the necessary calculations of a custom indicator must be placed in the predetermined function OnCalculate(). If you use a short form of the OnCalculate() function call, like

```
int OnCalculate (const int rates_total, const int prev_calculated, const int begin, co
```

then the *rates_total* variable contains the value of the total number of elements of the price[] array, passed as an input parameter for calculating indicator values.

Parameter *prev_calculated* is the result of the execution of OnCalculate() at the previous call; it allows organizing a saving algorithm for calculating indicator values. For example, if the current value rates_total = 1000, prev_calculated = 999, then perhaps it's enough to make calculations only for one value of each indicator buffer.

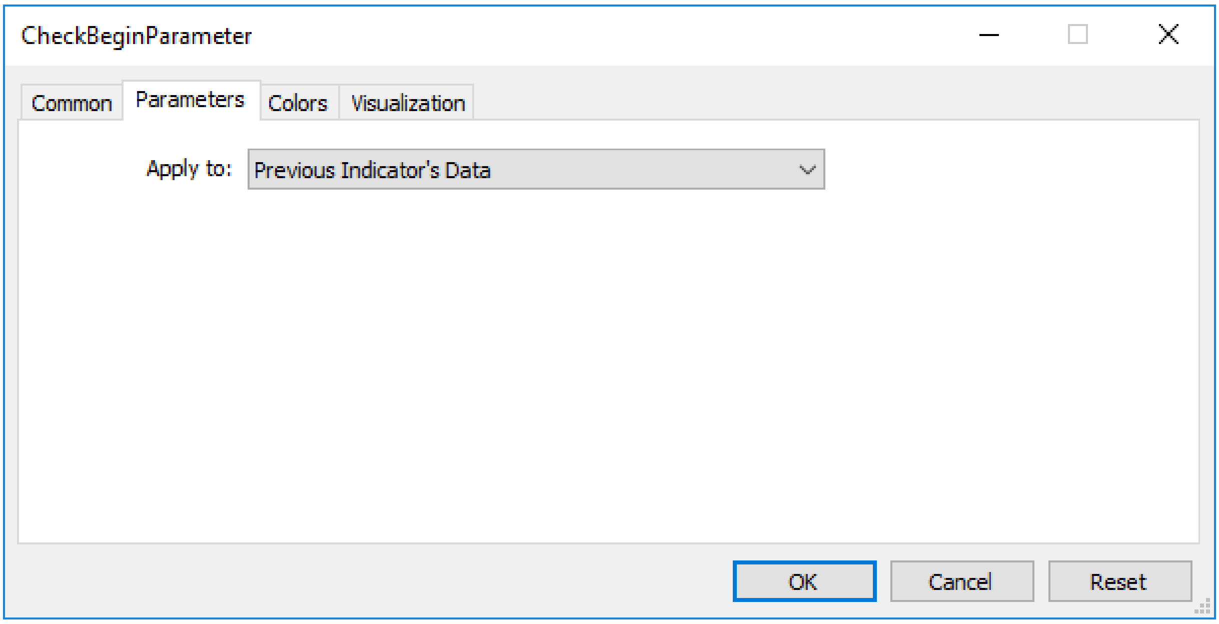
If the information about the size of the input array price would have been unavailable, then it would lead to the necessity to make calculations for 1000 values of each indicator buffer. At the first call of OnCalculate() value prev_calculated = 0. If the price[] array has changed somehow, then in this case prev_calculated is also equal to 0.

The *begin* parameter shows the number of initial values of the price array, which don't contain data for calculation. For example, if values of Accelerator Oscillator (for which the first 37 values aren't calculated) were used as an input parameter, then begin = 37. For example, let's consider a simple indicator:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Label1"
```

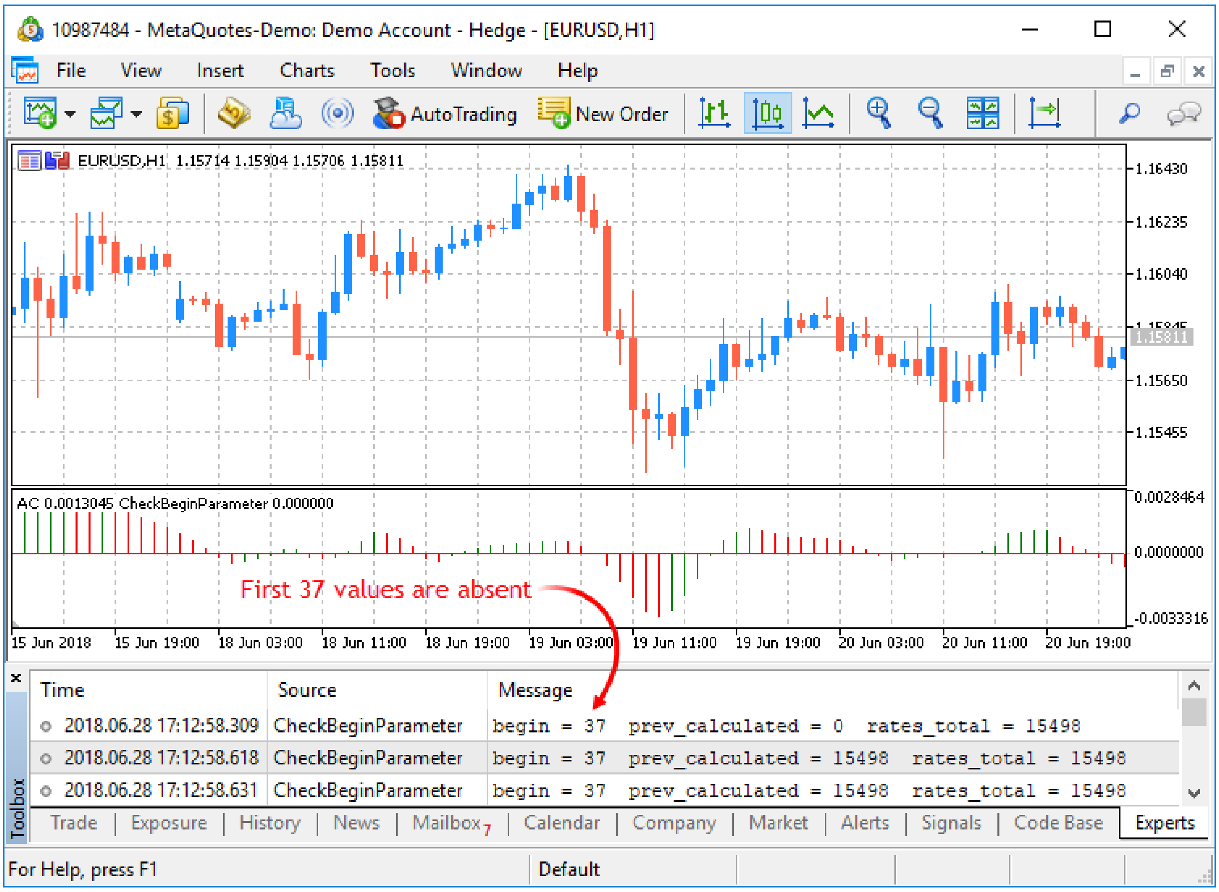
```
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         Label1Buffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])

  {
//---
   Print("begin = ",begin,"  prev_calculated = ",prev_calculated,"  rates_total = ",ra
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Drag it from the "Navigator" window to the window of the Accelerator Oscillator indicator and we indicate that calculations will be made based on the values of the previous indicator:



Apply to: Previous Indicator's Data

As a result, the first call of OnCalculate() the value of prev_calculated will be equal to zero, and in further calls it will be equal to the rates_total value (until the number of bars on the price chart increases).


10987484 - MetaQuotes-Demo: Demo Account - Hedge - [EURUSD,H1]
File
View
Insert
Charts
Tools
Window
Help
AutoTrading
New Order
EURUSD,H1 1.15714 1.15904 1.15706 1.15811
1.16430
1.16235
1.16040
1.15811
1.15650
1.15455
AC 0.0013045 CheckBeginParameter 0.000000
0.0028464
0.0000000
-0.0033316
First 37 values are absent
15 Jun 2018
15 Jun 19:00
18 Jun 03:00
18 Jun 11:00
18 Jun 19:00
19 Jun 03:00
19 Jun 11:00
19 Jun 19:00
20 Jun 03:00
20 Jun 11:00
20 Jun 19:00
Toolbox
Trade
Exposure
History
News
Mailbox 7
Calendar
Company
Market
Alerts
Signals
Code Base
Experts
For Help, press F1
Default

```
Time                     Source               Message
2018.06.28 17:12:58.309  CheckBeginParameter  begin = 37  prev_calculated = 0  rates_total = 15498
2018.06.28 17:12:58.618  CheckBeginParameter  begin = 37  prev_calculated = 15498  rates_total = 15498
2018.06.28 17:12:58.631  CheckBeginParameter  begin = 37  prev_calculated = 15498  rates_total = 15498
```

The value of the begin parameter will be exactly equal to the number of initial bars, for which the values of the Accelerator indicator aren't calculated according to the logic of this indicator. If we look at the source code of the custom indicator Accelerator.mq5, we'll see the following lines in the OnInit() function:

```
//--- sets first bar from which index will be drawn
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,37);
```

Using the function PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, empty_first_values), we set the number of non-existing first values in the zero indicator array of a custom indicator, which we don't need to accept for calculation (empty_first_values). Thus, we have mechanisms to:

1. set the number of initial values of an indicator, which shouldn't be used for calculations in another custom indicator;
2. get information on the number of first values to be ignored when you call another custom indicator, without going into the logic of its calculations.

Indicator Styles in Examples

The MetaTrader 5 Client Terminal includes 38 technical indicators that can be used in MQL5 programs using appropriate functions. But the main advantage of the MQL5 language is the ability to create custom indicators, which can then be used in Expert Advisors or simply applied on price charts for the purpose of technical analysis.

The entire set of indicators can be derived from several base drawing styles, known as plotting. Plotting denotes a way of displaying data, which the indicator calculates, stores and provides on request. There are seven such basic plotting types:

1. A line
2. A section (segment)
3. Histogram
4. Arrow (symbol)
5. A painted area (filled channel)
6. Bars
7. Japanese candlesticks

Each plotting requires one to five arrays of the double type, in which indicator values are stored. For the purpose of convenience, these arrays are associated with the indicator buffers. The number of buffers in an indicator must be declared in advance using compiler directives, for example:

```
#property indicator_buffers 3 // Number of buffers
#property indicator_plots   2 // number of plots
```

The number of buffers in the indicator is always greater than or equal to the number of plots in the indicator.

Since each basic plotting type can have color variation or construction specifics, the actual number of plotting types in the MQL5 is 18:

Plotting	Description	Value buffers	Color buffers
DRAW_NONE	Is not visually displayed in the chart, but the values of the corresponding buffer can be viewed in the Data Window	1	-

Plotting	Description	Value buffers	Color buffers
DRAW_LINE	A line is plotted on the values of the corresponding buffer (empty values in the buffer are undesirable)	1	-
DRAW_SECTION	Is drawn as line segments between the values of the corresponding buffer (usually has a lot of empty values)	1	-
DRAW_HISTOGRAM	Is drawn as a histogram from the zero line to the values of the corresponding buffer (may have empty values)	1	-

Plotting	Description	Value buffers	Color buffers
DRAW_HISTOGRAM2	Is drawn as a histogram based on two indicator buffers (may have empty values)	2	-
DRAW_ARROW	Is drawn as symbols (may have empty values)	1	-
DRAW_ZIGZAG	Similar to the style DRAW_SECTION, but unlike it, can plot vertical segments on one bar	2	-
DRAW_FILLING	Color fill between two lines. 2 values of the corresponding buffers are shown in the Data Window	2	-
DRAW_BARS	Is drawn as bars. 4 values	4	-

Plotting	Description	Value buffers	Color buffers
	of the corresponding buffers are shown in the Data Window		
DRAW_CANDLES	Drawn as Japanese candlesticks. 4 values of the corresponding buffers are shown in the Data Window	4	-
DRAW_COLOR_LINE	A line for which you can alternate colors on different bars or change its color at any time	1	1
DRAW_COLOR_SECTION	Similar to the style DRAW_SECTION, but the color of each section can be set individually; color can also	1	1

Plotting	Description	Value buffers	Color buffers
	be set dynamically		
DRAW_COLOR_HISTOGRAM	Similar to the style DRAW_HISTOGRAM, but each strip may have a different color, you can set the color dynamically	1	1
DRAW_COLOR_HISTOGRAM2	Similar to the style DRAW_HISTOGRAM2, but each strip may have a different color, you can set the color dynamically	2	1
DRAW_COLOR_ARROW	Similar to the style DRAW_ARROW, but each symbol can have its color. Color can be	1	1

Plotting	Description	Value buffers	Color buffers
	changed dynamically		
DRAW_COLOR_ZIGZAG	The DRAW_ZIGZAG style with the options of individual coloring of sections and dynamic color changing	2	1
DRAW_COLOR_BARS	The DRAW_BARS style with the options of individual coloring of bars and dynamic color changing	4	1
DRAW_COLOR_CANDLES	The DRAW_CANDLES style with the options of individual coloring of candlesticks and dynamic color changing	4	1

The difference between an indicator buffer and an array

In each indicator, on its global level, you should declare one or more arrays of the double type, which then must be used as an indicator buffer using the SetIndexBuffer() function. To draw indicator plots, only the values of the indicator buffers are used, any other arrays cannot be used for this purpose. In addition, buffer values are displayed in the Data Window.

An indicator buffer should be dynamic and does not require specification of the size – the size of the array used as the indicator buffer is set by the terminal execution subsystem automatically.

After the array is bound to the indicator buffer, the indexing direction is set by default like in ordinary arrays, but you can use the ArraySetAsSeries() function to change the way of access to the array elements. By default, the indicator buffer is used to store data used for plotting (INDICATOR_DATA).

If the calculation of indicator values requires holding intermediate calculations and storing the additional values for each bar, then such an array can be declared as a calculation buffer during binding (INDICATOR_CALCULATIONS). For the intermediate values, you can also use a regular array, but in this case, the programmer has to manage the size of the array.

Some plots allow setting a color for each bar. To store the information about color, color buffers are used (INDICATOR_COLOR_INDEX). The color is an integer type color, but all indicator buffers must be of type double. Values of color and auxiliary (INDICATOR_CALCULATIONS) buffers cannot be obtained by using CopyBuffer().

The number of indicator buffers must be specified using the compiler directive #property indicator_buffers number_of_buffers:

```
#property indicator_buffers 3  //  the indicator has 3 buffers
```

The maximum allowed number of buffers in one indicator is 512.

Relevance of Indicator Buffers and Plotting

Each plotting is based on one or more indicator buffers. So, for displaying simple candlesticks, four values are required - Open, High, Low and Close prices. Accordingly, to display an indicator in the form of candlesticks, it is necessary to declare 4 indicator buffers and 4 arrays of the double type for them. For example:

```
//--- The indicator has four indicator buffers
#property indicator_buffers 4
//--- The indicator has one plotting
#property indicator_plots   1
//--- Graphical plotting number 1 will appear as candlesticks
#property indicator_type1   DRAW_CANDLES
//--- Candlestick will be drawn in clrDodgerBlue
#property indicator_color1  clrDodgerBlue
//--- 4 arrays for the indicator buffers
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

Graphical plots automatically use indicator buffers in accordance with the plot number. Numbering of plots starts with 1, numbering of buffers starts with zero. If the first plotting requires 4 indicator buffers, then the first 4 indicator buffers will be used to draw it. These four buffers should be linked with the appropriate arrays with correct indexing using the SetIndexBuffer() function.

```
//--- Binding arrays with indicator buffers
   SetIndexBuffer(0,OBuffer,INDICATOR_DATA);  // The first buffer corresponds to the
   SetIndexBuffer(1,HBuffer,INDICATOR_DATA);  // The second buffer corresponds to inde
   SetIndexBuffer(2,LBuffer,INDICATOR_DATA);  // The third buffer corresponds to inde
   SetIndexBuffer(3,CBuffer,INDICATOR_DATA);  // The fourth buffer corresponds to inde
```

The plotting candlesticks, the indicator will use just the first four buffers, because plotting of "candlesticks" was announced under the first number.

Change the example, and add plotting of a simple line - DRAW_LINE. Now suppose that the line is numbered 1, and the candlesticks are number 2. The number of buffers and the number of plots has increased.

```
//--- The indicator has 5 indicator buffers
#property indicator_buffers 5
//--- The indicator has 2 plots
#property indicator_plots   2
//--- Plot 1 is a line
#property indicator_type1   DRAW_LINE
//--- The color of the line is clrDodgerRed
#property indicator_color1  clrDodgerRed
//--- Plot 2 is drawn as Japanese candlesticks
#property indicator_type2   DRAW_CANDLES
//--- The color of the candlesticks is clrDodgerBlue
#property indicator_color2  clrDodgerBlue
//--- 5 arrays for indicator buffers
double LineBuffer[];
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

The order of the plots has changed, and now the line comes first, followed by Japanese candlesticks. Therefore, the order of the buffers is appropriate - first we announce a buffer for the line with the zero index, and then four buffers for the candlesticks.

```
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);  // The first buffer corresponds to i
//--- Binding arrays with indicator buffers for the candlesticks
   SetIndexBuffer(1,OBuffer,INDICATOR_DATA);     // The second buffer corresponds to
   SetIndexBuffer(2,HBuffer,INDICATOR_DATA);     // The third buffer corresponds to i
   SetIndexBuffer(3,LBuffer,INDICATOR_DATA);     // The fourth buffer corresponds to
   SetIndexBuffer(4,CBuffer,INDICATOR_DATA);      // The fifth buffer corresponds to ir
```

The number of buffers and plots can be set only by using compiler directives, it is impossible to change these properties dynamically using functions.

Color Versions of Styles

As can be seen in the table, the styles are divided into two groups. The first group includes styles in whose name there is no word **COLOR**, we call these styles basic:

- DRAW_LINE
- DRAW_SECTION
- DRAW_HISTOGRAM
- DRAW_HISTOGRAM2
- DRAW_ARROW
- DRAW_ZIGZAG
- DRAW_FILLING
- DRAW_BARS
- DRAW_CANDLES

In the second group, the style names contain the word **COLOR**, let's call them color versions:

- DRAW_COLOR_LINE
- DRAW_COLOR_SECTION
- DRAW_COLOR_HISTOGRAM
- DRAW_COLOR_HISTOGRAM2
- DRAW_COLOR_ARROW
- DRAW_COLOR_ZIGZAG
- DRAW_COLOR_BARS
- DRAW_COLOR_CANDLES

All color versions of styles differ from the basic ones in that they allow specifying a color for each part of the plotting. The minimal part of plotting is a bar, so we can say that the color versions allow setting the color on each bar.

Exceptions are styles DRAW_NONE and DRAW_FILLING, they do not have color versions.

To set the plotting color on each bar, an additional buffer for storing the color index has been added to the color version. These indices indicate the number of a color in a special array, which contains a predefined set of colors. The size of the array of colors is 64. This means that each color version of a style allows painting a plot in 64 different colors.

The set and the number of colors in the special array of colors can be set via a compiler directive #property indicator_color, where you can specify all the necessary colors separated by commas. For example, such an entry in an indicator:

```
//--- Define 8 colors for coloring candlesticks (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
```

It states that for plotting 1, 8 colors are set, which will be placed in a special array. Further in the program we will not specify the color of the plotting, but only its index. If we want to set red color for the bar number **K**, the color index value from an array should be set in the color buffer of the indicator. The red color is specified first in the directive, it corresponds to the index number 0.

```
  //--- set the candlestick color clrRed
  col_buffer[buffer_index]=0;
```

The set of colors is not given once and for all, it can be changed dynamically using PlotIndexSetInteger(). Example:

```
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          color_array[i]);      //  A new color
```

Properties of the indicator and plotting

For indicator plots, properties can be set by means of compiler directives and using the appropriate functions. Read more information about this in Connection between Indicator Properties and Functions. Dynamic change of indicator properties using special functions allows creating more flexible custom indicators.

Start of Indicator Drawing on the Chart

In many cases, according to the conditions of the algorithm, it is impossible to start calculating the indicator values immediately with the current bar, since it is necessary to provide a minimum number of previous bars available in history. For example, many types of smoothing imply using an array of prices over the previous N bars, and on the basis of these values, the indicator value on the current bar is calculated.

In such cases, either there is no way to calculate the indicator values for the first N bars, or these values are not intended to be displayed on the chart and are only subsidiary for calculating further values. To avoid plotting of the indicator on the first N bars of the history, set the N value to the PLOT_DRAW_BEGIN property for the corresponding plot:

```
//--- Binding arrays with indicator buffers for the candlesticks
PlotIndexSetInteger(number_of_plot,PLOT_DRAW_BEGIN,N);
```

Here:

- number_of_plot - a value from zero to indicator_plots-1 (numbering of plots starts with zero).
- N - the number of first bars in the history, on which the indicator should not be displayed on the chart.

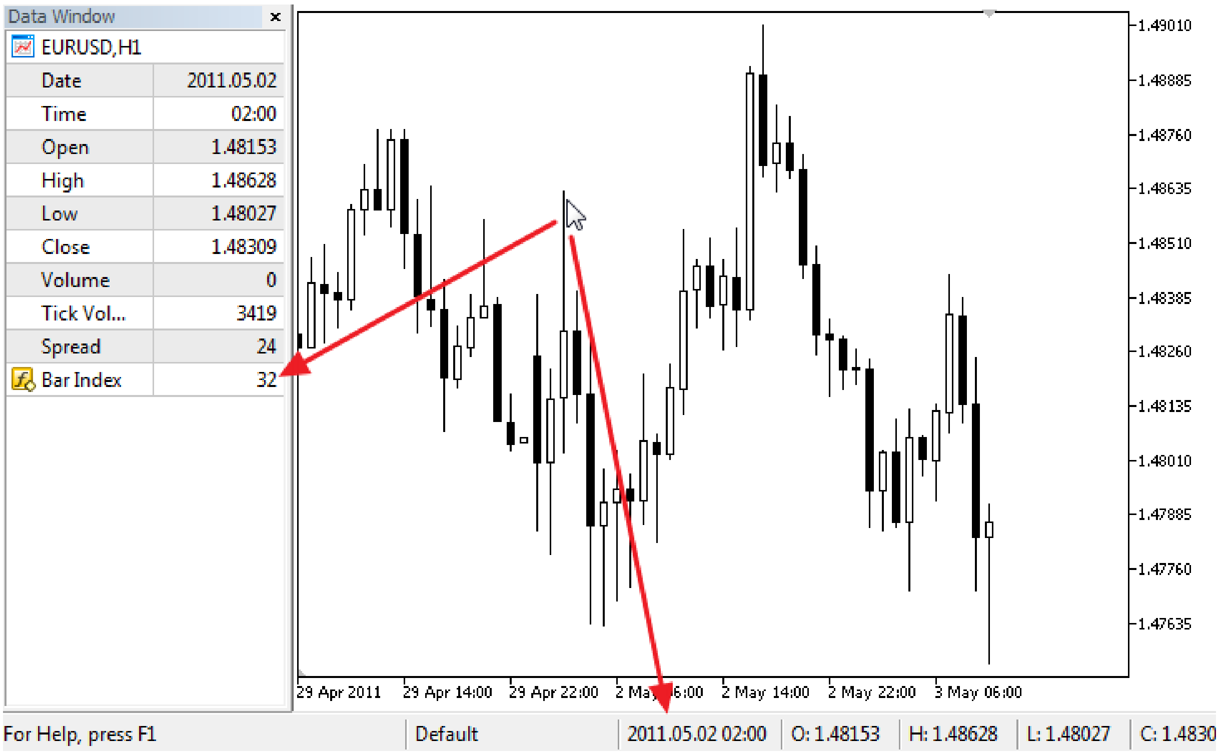
DRAW_NONE

The DRAW_NONE style is designed for use in cases where it is necessary to calculate the values of a buffer and show them in the Data Window, but plotting on the chart is not required. To set up the accuracy use the expression IndicatorSetInteger(INDICATOR_DIGITS,num_chars) in the OnInit() function:

```
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- Set the accuracy of values to be displayed in the Data Window
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
   return(INIT_SUCCEEDED);
  }
```

The number of buffers required for plotting DRAW_NONE is 1.

An example of the indicator that shows the number of the bar on which the mouse currently hovers in the Data Window. The numbering corresponds to the timeseries, meaning the current unfinished bar has the zero index, and the oldest bar has the largest index.



Data Window

EURUSD,H1

Date	2011.05.02
Time	02:00
Open	1.48153
High	1.48628
Low	1.48027
Close	1.48309
Volume	0
Tick Vol...	3419
Spread	24
Bar Index	32

For Help, press F1 | Default | 2011.05.02 02:00 | O: 1.48153 | H: 1.48628 | L: 1.48027 | C: 1.4830

1.49010
1.48885
1.48760
1.48635
1.48510
1.48385
1.48260
1.48135
1.48010
1.47885
1.47760
1.47635
29 Apr 2011
29 Apr 14:00
29 Apr 22:00
2 May 06:00
2 May 14:00
2 May 22:00
3 May 06:00

Note that despite the fact that, for red color is set plotting #1, the indicator does not draw anything on the chart.

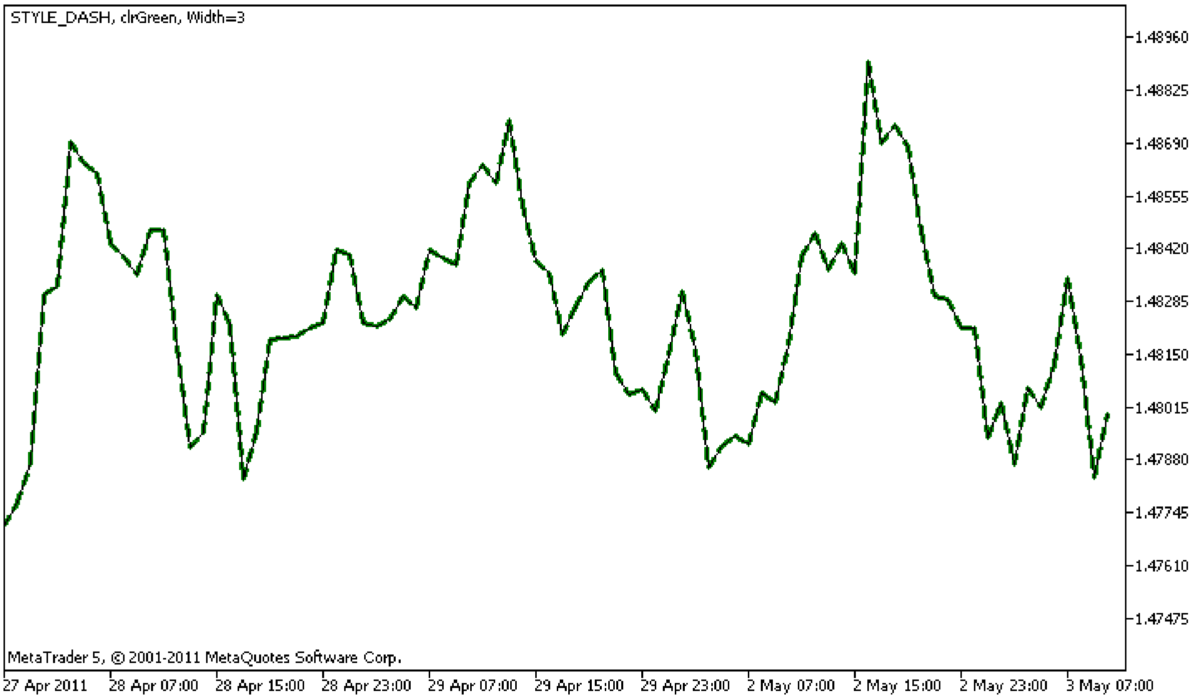
```
//+------------------------------------------------------------------+
//|                                                    DRAW_NONE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Invisible
#property indicator_label1  "Bar Index"
#property indicator_type1   DRAW_NONE
#property indicator_style1  STYLE_SOLID
#property indicator_color1  clrRed
#property indicator_width1  1
//--- indicator buffers
double         InvisibleBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- Set the accuracy of values to be displayed in the Data Window
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static datetime lastbar=0;
//--- If this is the first calculation of the indicator
   if(prev_calculated==0)
     {
      //--- Renumber the bars for the first time
      CalcValues(rates_total,close);
      //--- Remember the opening time of the current bar in lastbar
      lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
     }
   else
     {
      //--- If a new bar has appeared, its open time differs from lastbar
      if(lastbar!=SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE))
        {
         //--- Renumber the bars once again
         CalcValues(rates_total,close);
         //--- Update the opening time of the current bar in lastbar
         lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Number the bars like in a timeseries                             |
//+------------------------------------------------------------------+
void CalcValues(int total,double const  &array[])
  {
//--- Set indexing of the indicator buffer like in a timeseries
   ArraySetAsSeries(InvisibleBuffer,true);
//--- Fill in each bar with its number
   for(int i=0;i<total;i++) InvisibleBuffer[i]=i;
  }
```

DRAW_LINE

DRAW_LINE draws a line of the specified color by the values of the indicator buffer. The width, style and color of the line can be set using the compiler directives and dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The number of buffers required for plotting DRAW_LINE is 1.

An example of the indicator that draws a line using Close prices of bars. The line color, width and style change randomly every N=5 ticks.



Note that initially for **plot1** with DRAW_LINE the properties are set using the compiler directive #property, and then in the OnCalculate() function these three properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                                    DRAW_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_LINE"
#property description "It draws a line of a specified color at Close prices"
#property description "Color, width and style of lines is changed randomly"
#property description "after every N ticks"
```

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- Line properties are set using the compiler directives
#property indicator_label1  "Line"      // Name of a plot for the Data Window
#property indicator_type1   DRAW_LINE   // Type of plotting is line
#property indicator_color1  clrRed      // Line color
#property indicator_style1  STYLE_SOLID // Line style
#property indicator_width1  1           // Line Width
//--- input parameter
input int      N=5;         // Number of ticks to change
//--- An indicator buffer for the plot
double         LineBuffer[];
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//--- Initializing the generator of pseudo-random numbers
   MathSrand(GetTickCount());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Block for calculating indicator values
   for(int i=0;i<rates_total;i++)
     {
      LineBuffer[i]=close[i];
     }

//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of the drawn line in the indicator        |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the color of the line
//--- Get a random number
   int number=MathRand();
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+(string)colors[color_index];

//--- A block for changing the width of the line
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+", Width="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
```

```
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_SECTION

DRAW_SECTION draws sections of the specified color by the values of the indicator buffer. The width, color and style of the line can be specified like for the DRAW_LINE style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

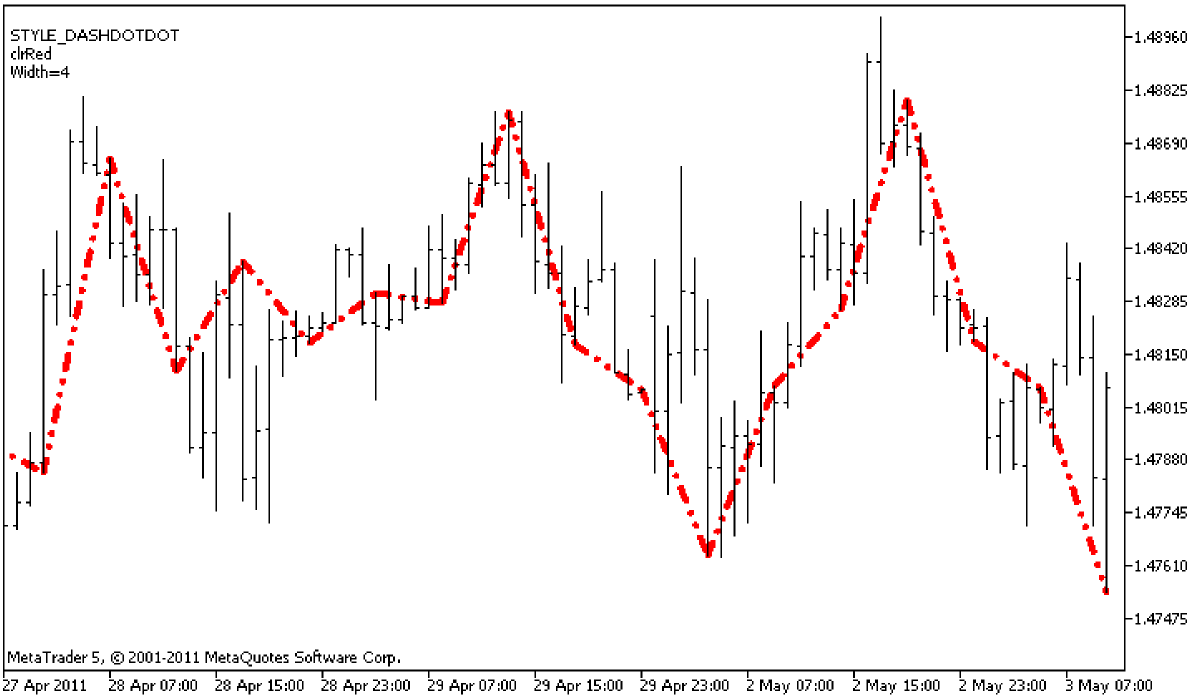
Sections are drawn from one non-empty value to another non-empty value of the indicator buffer, empty values are ignored. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property: For example, if the indicator should be drawn as a sequence of sections on non-zero values, then you need to set the zero value as an empty one:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to the elements that should not be plotted.

The number of buffers required for plotting DRAW_SECTION is 1.

An example of the indicator that draws sections between the High and Low prices. The color, width and style of **all** sections change randomly every **N** ticks.



Note that initially for **plot1** with DRAW_SECTION the properties are set using the compiler directive #property, and then in the OnCalculate() function these three properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                                 DRAW_SECTION.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_SECTION"
#property description "Draws straight sections every bars bars"
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Section
#property indicator_label1  "Section"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameter
input int      bars=5;           // The length of sections in bars
input int      N=5;              // The number of ticks to change the style of sectio
//--- An indicator buffer for the plot
double         SectionBuffer[];
//--- An auxiliary variable to calculate ends of sections
int            divider;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,SectionBuffer,INDICATOR_DATA);
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- Check the indicator parameter
   if(bars<=0)
     {
      PrintFormat("Invalid value of parameter bar=%d",bars);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=2*bars;
//---+
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- The number of the bar from which the calculation of indicator values starts
   int start=0;
//--- If the indicator has been calculated before, then set start on the previous bar
   if(prev_calculated>0) start=prev_calculated-1;
//--- Here are all the calculations of the indicator values
   for(int i=start;i<rates_total;i++)
     {
      //--- Get a remainder of the division of the bar number by 2*bars
      int rest=i%divider;
      //--- If the bar number is divisible by 2*bars
      if(rest==0)
        {
         //--- Set the end of the section at the High price of this bar
         SectionBuffer[i]=high[i];
        }
      //---If the remainder of the division is equal to bars,
      else
        {
         //--- Set the end of the section at the High price of this bar
         if(rest==bars) SectionBuffer[i]=low[i];
         //--- If nothing happened, ignore the bar - set 0
         else SectionBuffer[i]=0;
        }
      }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of sections in the indicator              |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block of line color change
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the width
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the line style
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

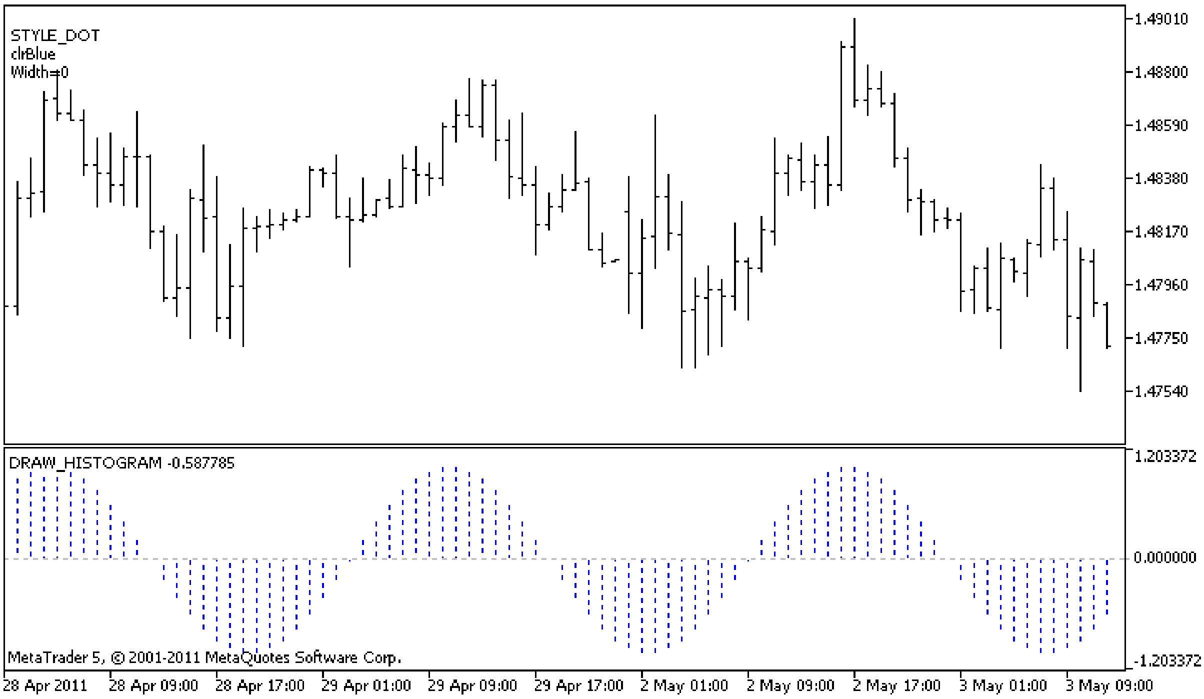
DRAW_HISTOGRAM

The DRAW_HISTOGRAM style draws a histogram as a sequence of columns of a specified color from zero to a specified value. Values are taken from the indicator buffer. The width, color and style of the column can be specified like for the DRAW_LINE style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

Since a column from the zero level is drawn on each bar, DRAW_HISTOGRAM should better be used in a separate chart window. Most often this type of plotting is used to create indicators of the oscillator type, for example, Bears Power or OsMA. For the empty non-displayable values the zero value should be specified.

The number of buffers required for plotting DRAW_HISTOGRAM is 1.

An example of the indicator that draws a sinusoid of a specified color based on the MathSin() function. The color, width and style of **all** histogram columns change randomly each **N** ticks. The bars parameter specifies the period of the sinusoid, that is after the specified number of bars the sinusoid will repeat the cycle.



Note that initially for **plot1** with DRAW_HISTOGRAM the properties are set using the compiler directive #property, and then in the OnCalculate() function these three properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                           DRAW_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
```

```
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars in the cycle of the

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=30;          // The period of a sinusoid in bars
input int      N=5;              // The number of ticks to change the histogram
//--- indicator buffers
double         HistogramBuffer[];
//--- A factor to get the 2Pi angle in radians, when multiplied by the bars parameter
double    multiplier;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- Calculate the multiplier
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Set the value of bars=%d greater than 1",bars);
      //--- Early termination of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Calculate the indicator values
   int start=0;
//--- If already calculated during the previous starts of OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculat
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      HistogramBuffer[i]=sin(i*multiplier);
     }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of lines in the indicator                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the color of the line
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the width
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the line style
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_HISTOGRAM2

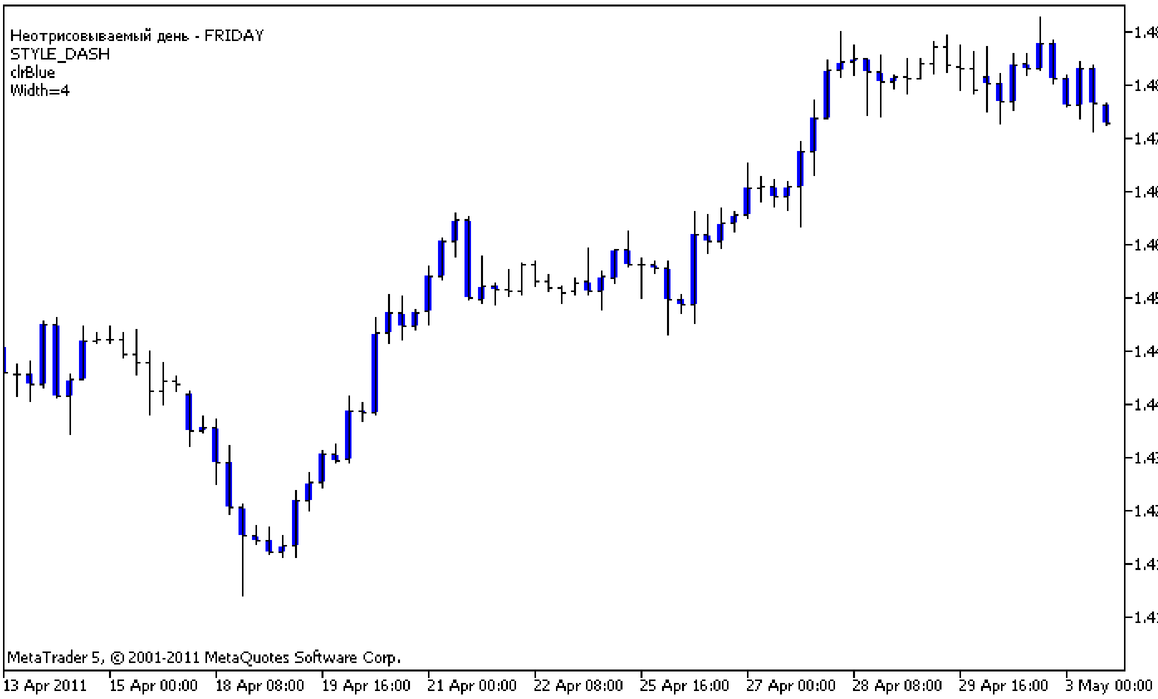
The DRAW_HISTOGRAM2 style draws a histogram of a specified color – vertical segments using the values of two indicator buffers. The width, color and style of the segments can be specified like for the DRAW_LINE style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

The DRAW_HISTOGRAM2 style can be used in a separate subwindow of a chart and in its main window. For empty values nothing is drawn, all the values in the indicator buffers need to be set explicitly. Buffers are not initialized with a zero value.

The number of buffers required for plotting DRAW_HISTOGRAM2 is 2.

An example of the indicator that plots a vertical segment of the specified color and width between the Open and Close prices of each bar. The color, width and style of **all** histogram columns change randomly each **N** ticks. During the start of the indicator, in the OnInit() function, the number of the day of week for which the histogram will not be drawn - invisible_day - is set randomly. For this purpose an empty value is set PLOT_EMPTY_VALUE=0:

```
//--- Set an empty value
   PlotIndexSetDouble(index_of_plot_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```



Note that initially for **plot1** with DRAW_HISTOGRAM2 the properties are set using the compiler directive #property, and then in the OnCalculate() function these three properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                               DRAW_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot Histogram_2
#property indicator_label1  "Histogram_2"
#property indicator_type1   DRAW_HISTOGRAM2
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;              // The number of ticks to change the histogram
//--- indicator buffers
double         Histogram_2Buffer1[];
double         Histogram_2Buffer2[];
//--- The day of the week for which the indicator is not plotted
int invisible_day;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Histogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Histogram_2Buffer2,INDICATOR_DATA);
//--- Set an empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- Get a random number from 0 to 5
   invisible_day=MathRand()%6;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Calculate the indicator values
   int start=0;
//--- To get the day of week by the open price of each bar
   MqlDateTime dt;
//--- If already calculated during the previous starts of OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculat
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
      if(dt.day_of_week==invisible_day)
        {
         Histogram_2Buffer1[i]=0;
         Histogram_2Buffer2[i]=0;
        }
      else
        {
         Histogram_2Buffer1[i]=open[i];
         Histogram_2Buffer2[i]=close[i];
        }
     }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of lines in the indicator                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block of line color change
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the line width
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the line style
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- Add information about the day that is omitted in calculations
   comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_ARROW

The DRAW_ARROW style draws arrows of the specified color (symbols of the set Wingdings) based on the value of the indicator buffer. The width and color of the symbols can be specified like for the DRAW_LINE style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of an indicator based on the current situation.

The symbol code is set using the PLOT_ARROW property.

```
//--- Define the symbol code from the Wingdings font to draw in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

The default value of PLOT_ARROW=159 (a circle).

Each arrow is actually a symbol that has the height and the anchor point, and can cover some important information on a chart (for example, the closing price at the bar). Therefore, we can additionally specify the vertical shift in pixels, which does not depend on the scale of the chart. The arrows will be shifted down by the specified number of pixels, although the values of the indicator will remain the same:

```
//--- Set the vertical shift of arrows in pixels
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

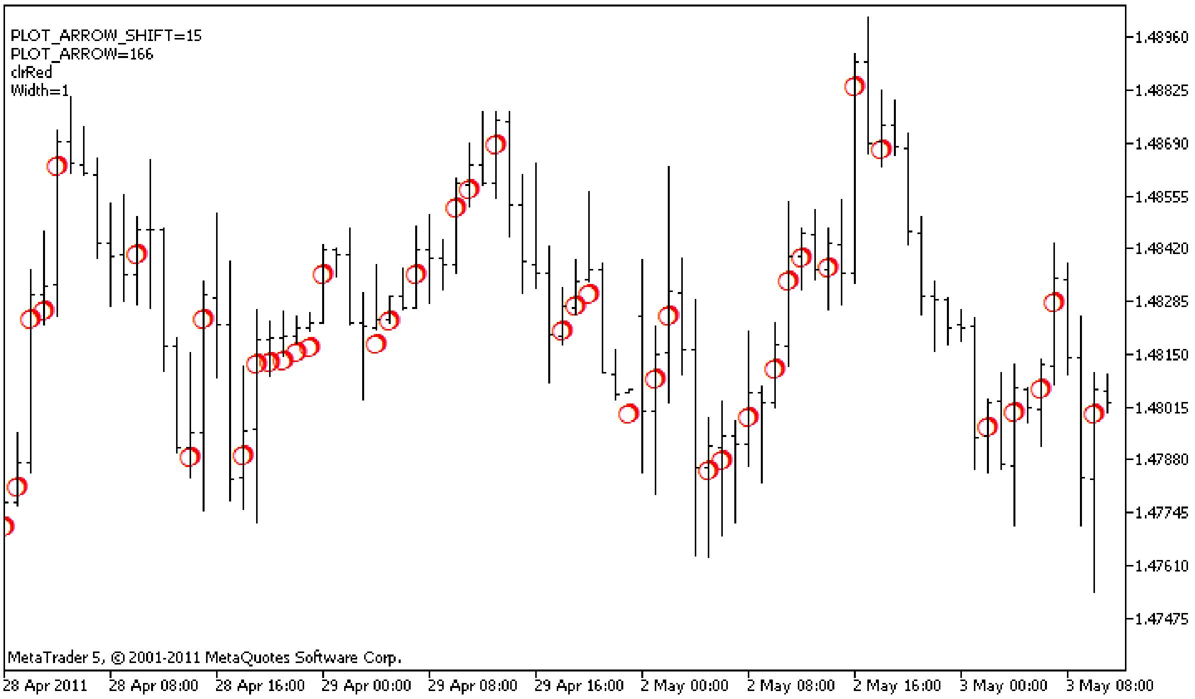
A negative value of PLOT_ARROW_SHIFT means the shift of arrows upwards, a positive values shifts the arrow down.

The DRAW_ARROW style can be used in a separate subwindow of a chart and in its main window. Empty values are not drawn and do not appear in the "Data Window", all the values in the indicator buffers should be set explicitly. Buffers are not initialized with a zero value.

```
//--- Set an empty value
   PlotIndexSetDouble(index_of_plot_DRAW_ARROW,PLOT_EMPTY_VALUE,0);
```

The number of buffers required for plotting DRAW_ARROW is 1.

An example of the indicator, which draws arrows on each bar with the close price higher than the close price of the previous bar. The color, width, shift and symbol code of **all** arrows are changed randomly every **N** ticks.



In the example, for **plot1** with the DRAW_ARROW style, the properties, color and size are specified using the compiler directive #property, and then in the OnCalculate() function the properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                                   DRAW_ARROW.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ARROW"
#property description "Draws arrows set by Unicode characters, on a chart"
#property description "The color, size, shift and symbol code of the arrow are change
#property description "after every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Arrows
#property indicator_label1  "Arrows"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrGreen
#property indicator_width1  1
//--- input parameters
input int      N=5;         // Number of ticks to change
input ushort   code=159;    // Symbol code to draw in DRAW_ARROW
//--- An indicator buffer for the plot
double         ArrowsBuffer[];
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ArrowsBuffer,INDICATOR_DATA);
//--- Define the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- Set the vertical shift of arrows in pixels
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- Set as an empty value 0
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the color, size, shift and code of the arrow
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Block for calculating indicator values
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- Calculation loop
   for(int i=1;i<rates_total;i++)
     {
      //--- If the current Close price is higher than the previous one, draw an arrow
      if(close[i]>close[i-1])
         ArrowsBuffer[i]=close[i];
      //--- Otherwise specify the zero value
      else
         ArrowsBuffer[i]=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Change the appearance of symbols in the indicator                |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the indicator properties
   string comm="";
//--- A block for changing the arrow color
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the size arrows
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The size is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the arrow size
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the arrow code (PLOT_ARROW)
   number=MathRand();
//--- Get the remainder of integer division to calculate a new code of the arrow (fro
   int code_add=number%20;
//--- Set the new symbol code as the result of code+code_add
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- Write the symbol code PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;
//--- A block for changing the vertical shift of arrows in pixels
   number=MathRand();
//--- Get the shift as the remainder of the integer division
   int shift=20-number%41;
//--- Set the new shift from -20 to 20
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- Write the shift PLOT_ARROW_SHIFT
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_ZIGZAG

The DRAW_ZIGZAG style draws segments of a specified color based on the values of two indicator buffers. This style is very similar to DRAW_SECTION, but unlike the latter, it allows drawing vertical segments within one bar, if values of both indicator buffers are set for this bar. The segments are plotted from a value in the first buffer to a value in the second indicator buffer. None of the buffers can contain only empty values, since in this case nothing is plotted.

The width, color and style of the line can be specified like for the DRAW_SECTION style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

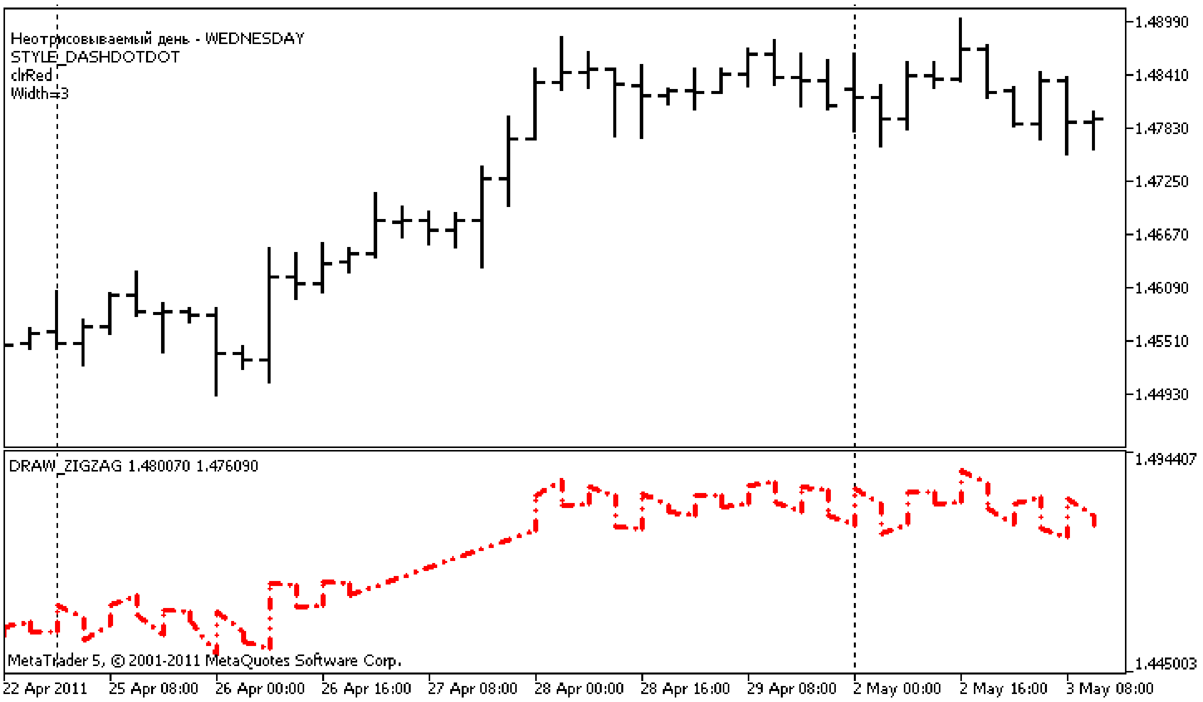
Sections are drawn from a non-empty value of one buffer to a non-empty value of another indicator buffer. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_ZIGZAG is 2.

An example of the indicator that plots a saw based on the High and Low prices. The color, width and style of the zigzag lines change randomly every N ticks.


Неотрисовываемый день - WEDNESDAY
STYLE_DASHDOTDOT
clrRed
Width=3
1.48990
1.48410
1.47830
1.47250
1.46670
1.46090
1.45510
1.44930
DRAW_ZIGZAG 1.480070 1.476090
1.494407
1.445003
MetaTrader 5, © 2001-2011 MetaQuotes Software Corp.
22 Apr 2011
25 Apr 08:00
26 Apr 00:00
26 Apr 16:00
27 Apr 08:00
28 Apr 00:00
28 Apr 16:00
29 Apr 08:00
2 May 00:00
2 May 16:00
3 May 08:00

Note that initially for **plot1** with DRAW_ZIGZAG the properties are set using the compiler directive #property, and then in the OnCalculate() function these properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                                  DRAW_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_ZIGZAG"
#property description "It draws a \"saw\" as straight segments, skipping the bars of 
#property description "The day to skip is selected randomly during indicator start"
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ZigZag
#property indicator_label1  "ZigZag"
#property indicator_type1   DRAW_ZIGZAG
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;              // Number of ticks to change
//--- indicator buffers
double         ZigZagBuffer1[];
double         ZigZagBuffer2[];
//--- The day of the week for which the indicator is not plotted
int invisible_day;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen};
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Binding arrays and indicator buffers
   SetIndexBuffer(0,ZigZagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ZigZagBuffer2,INDICATOR_DATA);
//--- Get a random value from 0 to 6, for this day the indicator is not plotted
   invisible_day=MathRand()%6;
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetString(0,PLOT_LABEL,"ZigZag1;ZigZag2");
```

```
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- The structure of time is required to get the day of week of each bar
   MqlDateTime dt;

//--- The start position of calculations
   int start=0;
//--- If the indicator was calculated on the previous tick, then start the calculation
   if(prev_calculated!=0) start=prev_calculated-1;
//--- Calculation loop
   for(int i=start;i<rates_total;i++)
     {
      //--- Write the bar open time in the structure
      TimeToStruct(time[i],dt);
      //--- If the day of the week of this bar is equal to invisible_day
      if(dt.day_of_week==invisible_day)
        {
         //--- Write empty values to buffers for this bar
         ZigZagBuffer1[i]=0;
         ZigZagBuffer2[i]=0;
        }
      //--- If the day of the week is ok, fill in the buffers
      else
        {
         //--- If the bar number if even
         if(i%2==0)
           {
            //---  Write High in the 1st buffer and Low in the 2nd one
            ZigZagBuffer1[i]=high[i];
            ZigZagBuffer2[i]=low[i];
           }
         //--- The bar number is odd
         else
           {
            //--- Fill in the bar in a reverse order
            ZigZagBuffer1[i]=low[i];
            ZigZagBuffer2[i]=high[i];
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of the zigzag segments                    |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the ZigZag properties
   string comm="";
//--- A block for changing the color of the ZigZag
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of the line
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
```

```
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- Add information about the day that is omitted in calculations
   comm="\r\nNot plotted day - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_FILLING

The DRAW_FILLING style plots a colored area between the values of two indicator buffers. In fact, this style draws two lines and fills the space between them with one of two specified colors. It is used for creating indicators that draw channels. None of the buffers can contain only empty values, since in this case nothing is plotted.

You can set two fill colors:

- the first color is used for the areas where values in the first buffer are greater than the values in the second indicator buffer;
- the second color is used for the areas where values in the second buffer are greater than the values in the first indicator buffer.

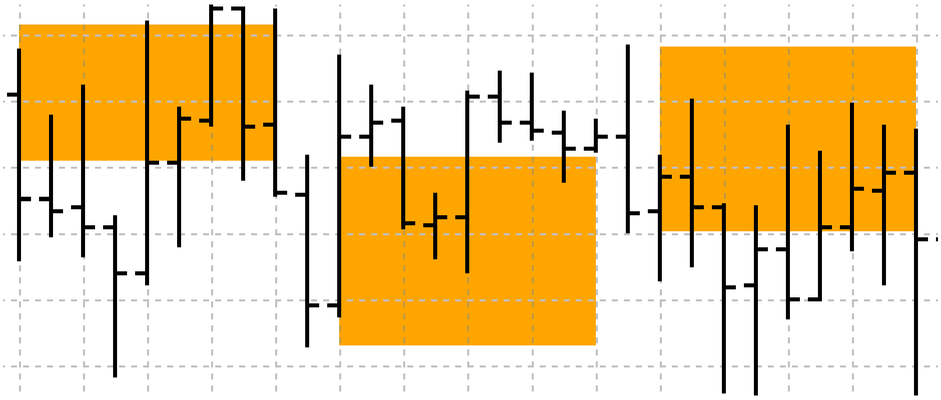
The fill color can be set using the compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is calculated for all bars, for which the values of the both indicator buffers are equal neither to 0 nor to the empty value. To specify what value should be considered as "empty", set this value in PLOT_EMPTY_VALUE property:

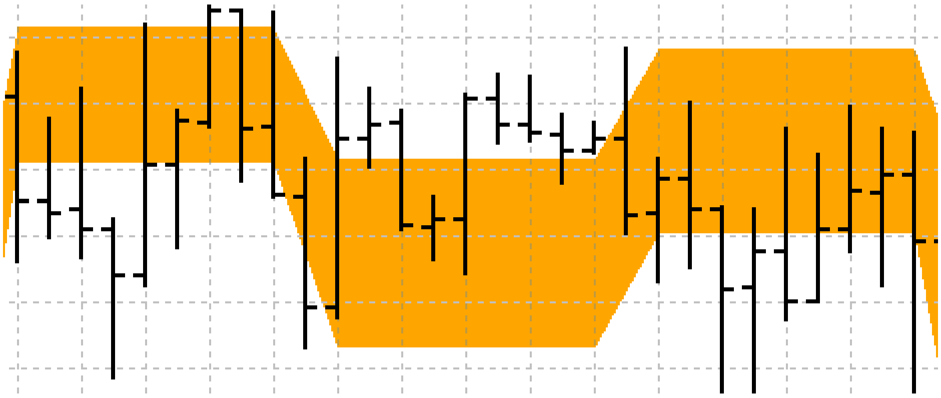
```
   #define INDICATOR_EMPTY_VALUE -1.0
   ...
//--- INDICATOR_EMPTY_VALUE (empty value) will not participate in calculation of
   PlotIndexSetDouble (DRAW_FILLING_creation_index,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VA
```

Drawing on the bars that do not participate in the indicator calculation will depend on the values in the indicator buffers:

- Bars, for which the values of both indicator buffers are equal to 0, do not participate in drawing the indicator. It means that the area with zero values is not filled out.



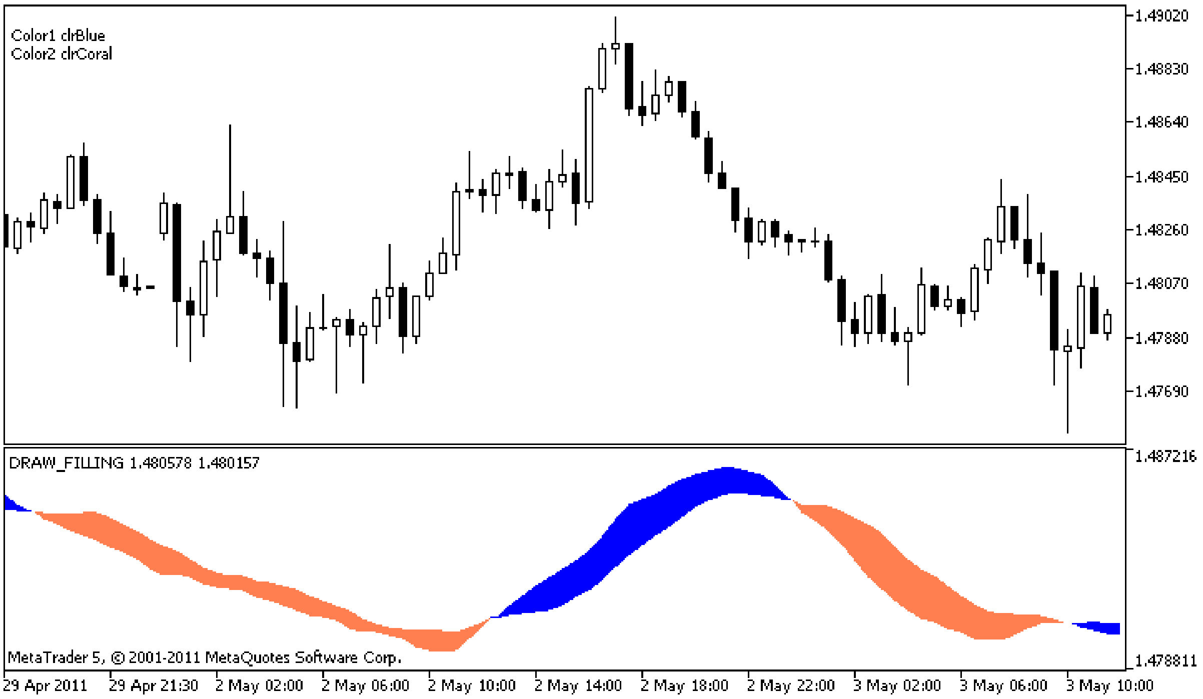
- Bars, for which the values of the indicator buffers are equal to the "empty value", participate in drawing the indicator. The area with empty values will be filled out so that to connect the areas with significant values.



It should be noted that if the "empty value" is equal to zero, the bars that do not participate in the indicator calculation are also filled out.

The number of buffers required for plotting DRAW_FILLING is 2.

An example of the indicator that draws a channel between two MAs with different averaging periods in a separate window. The change of the colors at the crossing of moving averages visually shows the change of the upward and downward trends. The colors change randomly every **N** ticks. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).


Color1 clrBlue
Color2 clrCoral
1.49020
1.48830
1.48640
1.48450
1.48260
1.48070
1.47880
1.47690
DRAW_FILLING 1.480578 1.480157
1.487216
1.478811
MetaTrader 5, © 2001-2011 MetaQuotes Software Corp.
29 Apr 2011
29 Apr 21:30
2 May 02:00
2 May 06:00
2 May 10:00
2 May 14:00
2 May 18:00
2 May 22:00
3 May 02:00
3 May 06:00
3 May 10:00

Note that initially for **plot1** with DRAW_FILLING the properties are set using the compiler directive #property, and then in the OnCalculate() function new colors are set randomly.

```
//+------------------------------------------------------------------+
//|                                                 DRAW_FILLING.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
```

```
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_FILLING"
#property description "It draws a channel between two MAs in a separate window"
#property description "The fill color is changed randomly"
#property description "after every N ticks"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot Intersection
#property indicator_label1  "Intersection"
#property indicator_type1   DRAW_FILLING
#property indicator_color1  clrRed,clrBlue
#property indicator_width1  1
//--- input parameters
input int      Fast=13;          // The period of a fast MA
input int      Slow=21;          // The period of a slow MA
input int      shift=1;          // A shift of MAs towards the future (positive)
input int      N=5;              // Number of ticks to change
//--- Indicator buffers
double         IntersectionBuffer1[];
double         IntersectionBuffer2[];
int fast_handle;
int slow_handle;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrAquamarine,clrBlanchedAlmond,clrBrown,clrCo
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,IntersectionBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,IntersectionBuffer2,INDICATOR_DATA);
//---
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//---
   fast_handle=iMA(_Symbol,_Period,Fast,0,MODE_SMA,PRICE_CLOSE);
   slow_handle=iMA(_Symbol,_Period,Slow,0,MODE_SMA,PRICE_CLOSE);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Make the first calculation of the indicator, or data has changed and requires a
   if(prev_calculated==0)
     {
      //--- Copy all the values of the indicators to the appropriate buffers
      int copied1=CopyBuffer(fast_handle,0,0,rates_total,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,rates_total,IntersectionBuffer2);
     }
   else // Fill only those data that are updated
     {
      //--- Get the difference in bars between the current and previous start of OnCal
      int to_copy=rates_total-prev_calculated;
      //--- If there is no difference, we still copy one value - on the zero bar
      if(to_copy==0) to_copy=1;
      //--- copy to_copy values to the very end of indicator buffers
      int copied1=CopyBuffer(fast_handle,0,0,to_copy,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,to_copy,IntersectionBuffer2);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the colors of the channel filling                        |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the color of the line
```

```
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);

//--- Get the index to select a new color as the remainder of integer division
   int color_index1=number%size;
//--- Set the first color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,colors[color_index1]);
//--- Write the first color
   comm=comm+"\r\nColor1 "+(string)colors[color_index1];

//--- Get the index to select a new color as the remainder of integer division
   number=MathRand(); // Get a random number
   int color_index2=number%size;
//--- Set the second color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,colors[color_index2]);
//--- Write the second color
   comm=comm+"\r\nColor2 "+(string)colors[color_index2];
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_BARS

The DRAW_BARS style draws bars on the values of four indicator buffers, which contain the Open, High, Low and Close prices. It is used for creating custom indicators as bars, including those in a separate subwindow of a chart and on other financial instruments.

The color of bars can be set using the compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

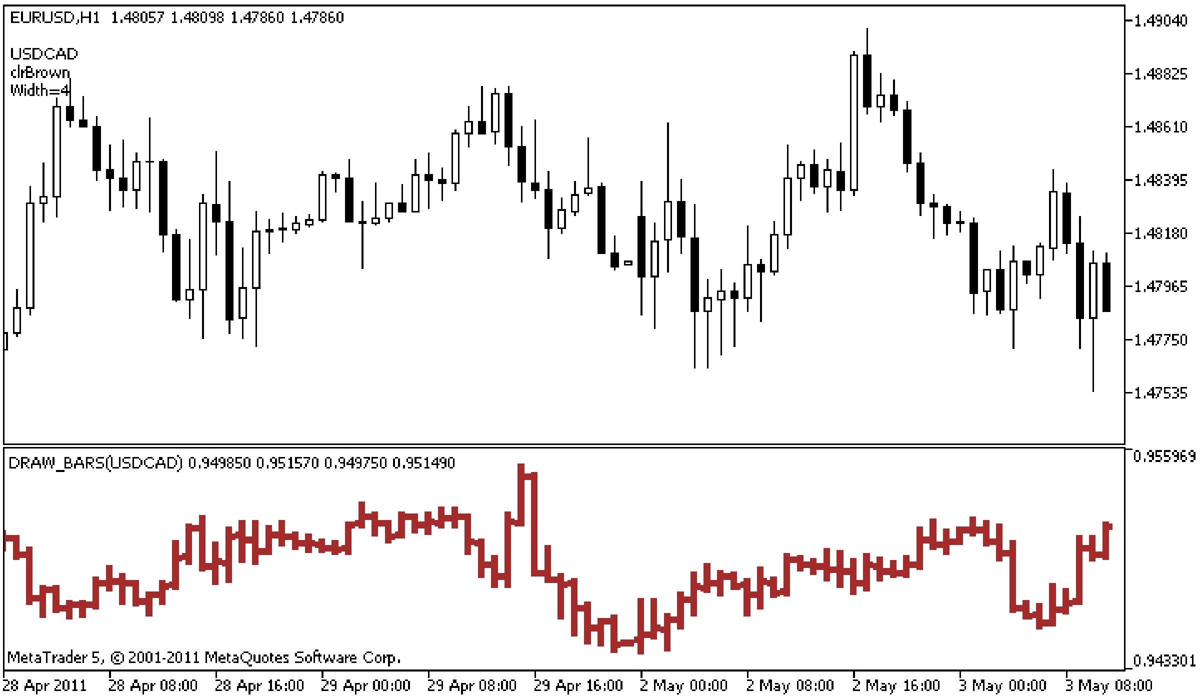
The indicator is drawn only to those bars, for which non-empty values of **all** four indicator buffers are set. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_BARS,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_BARS is 4. All buffers for the plotting should go one after the other in the given order: Open, High, Low and Close. None of the buffers can contain only empty values, since in this case nothing is plotted.

An example of the indicator that draws bars on a selected financial instrument in a separate window. The color of bars changes randomly every **N** ticks. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).


EURUSD,H1 1.48057 1.48098 1.47860 1.47860
USDCAD
clrBrown
Width=4
1.49040
1.48825
1.48610
1.48395
1.48180
1.47965
1.47750
1.47535
DRAW_BARS(USDCAD) 0.949850 0.951570 0.949750 0.951490
0.955969
0.943301
MetaTrader 5, © 2001-2011 MetaQuotes Software Corp.
28 Apr 2011
28 Apr 08:00
28 Apr 16:00
29 Apr 00:00
29 Apr 08:00
29 Apr 16:00
2 May 00:00
2 May 08:00
2 May 16:00
3 May 00:00
3 May 08:00

Please note that for **plot1** with the DRAW_BARS style, the color is set using the compiler directive #property, and then in the OnCalculate() function the color is set randomly from an earlier prepared list.

```
//+------------------------------------------------------------------+
//|                                                    DRAW_BARS.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_BARS"
#property description "It draws bars of a selected symbol in a separate window"
#property description "The color and width of bars, as well as the symbol are changed
#property description "every N ticks"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- plot Bars
#property indicator_label1  "Bars"
#property indicator_type1   DRAW_BARS
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;              // The number of ticks to change the type
input int      bars=500;         // The number of bars to show
input bool     messages=false;   // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double         BarsBuffer1[];
double         BarsBuffer2[];
double         BarsBuffer3[];
double         BarsBuffer4[];
//--- Symbol name
string symbol;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- If bars is very small - complete the work ahead of time
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,BarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,BarsBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,BarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,BarsBuffer4,INDICATOR_DATA);
//--- The name of the symbol, for which the bars are drawn
   symbol=_Symbol;
//--- Set the display of the symbol
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+symbol+")");
//--- An empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Select a new symbol from the Market watch window
      symbol=GetRandomSymbolName();
      //--- Change the line properties
      ChangeLineAppearance();

      int tries=0;
      //--- Make 5 attempts to fill in the buffers with the prices from symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total) && tries<5)
        {
         //--- A counter of calls of the CopyFromSymbolToBuffers() function
         tries++;
        }
      //--- Reset the counter of ticks to zero
      ticks=0;
     }
//--- return value of prev_calculated for next call
```

```
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Fill in the indicator buffers with prices                        |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total)
  {
//--- In the rates[] array, we will copy Open, High, Low and Close
   MqlRates rates[];
//--- The counter of attempts
   int attempts=0;
//--- How much has been copied
   int copied=0;
//--- Make 25 attempts to get a timeseries on the desired symbol
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- If failed to copy a sufficient number of bars
   if(copied!=bars)
     {
      //--- Form a message string
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- Show a message in a comment in the main chart window
      Comment(comm);
      //--- Show the message
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- Set the display of the symbol
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+name+")");
     }
//--- Initialize buffers with empty values
   ArrayInitialize(BarsBuffer1,0.0);
   ArrayInitialize(BarsBuffer2,0.0);
   ArrayInitialize(BarsBuffer3,0.0);
   ArrayInitialize(BarsBuffer4,0.0);
//--- Copy prices to the buffers
   for(int i=0;i<copied;i++)
     {
      //--- Calculate the appropriate index for the buffers
      int buffer_index=total-copied+i;
      //--- Write the prices to the buffers
      BarsBuffer1[buffer_index]=rates[i].open;
      BarsBuffer2[buffer_index]=rates[i].high;
      BarsBuffer3[buffer_index]=rates[i].low;
      BarsBuffer4[buffer_index]=rates[i].close;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| Randomly returns a symbol from the Market Watch                  |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- The number of symbols shown in the Market watch window
   int symbols=SymbolsTotal(true);
//--- The position of a symbol in the list - a random number from 0 to symbols
   int number=MathRand()%symbols;
//--- Return the name of a symbol at the specified position
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of bars                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the bar properties
   string comm="";
//--- A block for changing the color of bars
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the line color
   comm=comm+"\r\n"+(string)colors[color_index];

//--- A block for changing the width of bars
   number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);
```

```
//--- Write the symbol name
   comm="\r\n"+symbol+comm;

//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_CANDLES

The DRAW_CANDLES style draws candlesticks on the values of four indicator buffers, which contain the Open, High, Low and Close prices. It is used for creating custom indicators as a sequence of candlesticks, including those in a separate subwindow of a chart and on other financial instruments.

The color of candlesticks can be set using the compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of **all** four indicator buffers are set. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_CANDLES,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_CANDLES is 4. All buffers for the plotting should go one after the other in the given order: Open, High, Low and Close. None of the buffers can contain only empty values, since in this case nothing is plotted.

You can set up to three colors for the DRAW_CANDLES style affecting the candle look. If only one color is set, it is applied to all candles on a chart.

```
//--- identical candles with a single color applied to them
#property indicator_label1  "One color candles"
#property indicator_type1   DRAW_CANDLES
//--- only one color is specified, therefore all candles are of the same color
#property indicator_color1  clrGreen
```

If two comma-separated colors are specified, the first one is applied to candle outlines, while the second one is applied to the body.

```
//--- different colors for candles and wicks
#property indicator_label1  "Two color candles"
#property indicator_type1   DRAW_CANDLES
//--- green is applied to wicks and outlines, while white is applied to the body
#property indicator_color1  clrGreen,clrWhite
```

Specify three comma-separated colors so that rising and falling candles are displayed differently. In that case, the first color is applied to the candle outlines, while the second and third ones – to bullish and bearish candles.

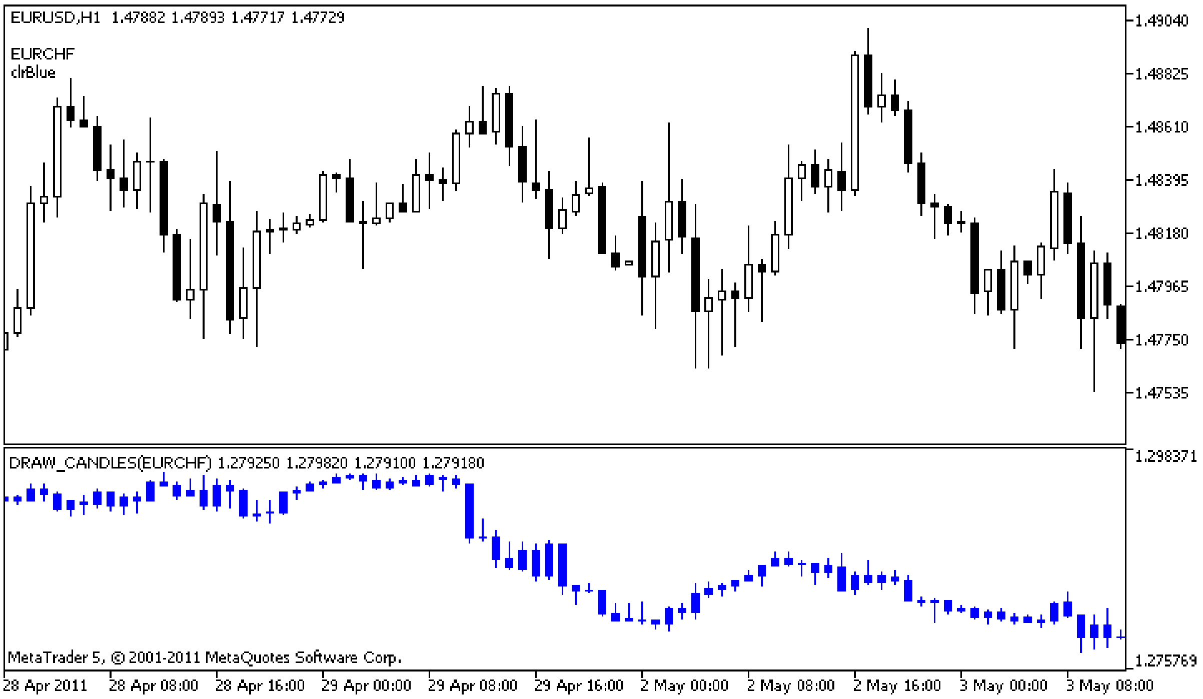
```
//--- different colors for candles and wicks
#property indicator_label1  "One color candles"
#property indicator_type1   DRAW_CANDLES
//--- wicks and outlines are green, bullish candle body is white, while bearish candle
#property indicator_color1  clrGreen,clrWhite,clrRed
```

Thus, the DRAW_CANDLES style allows you to create custom candle coloring options. Besides, all colors can be changed dynamically during the indicator operation using the PlotIndexSetInteger function (composition_index_DRAW_CANDLES, PLOT_LINE_COLOR, modifier_index, color), where modifier_index may have the following values:

- 0 – colors of outlines and wicks
- 1– bullish candle body color
- 2 – bearish candle body color

```
//--- set the color of outlines and wicks
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrBlue);
//--- set the bullish body color
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrGreen);
//--- set the bearish body color
PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrRed);
```

An example of the indicator that draws candlesticks for a selected financial instrument in a separate window. The color of candlesticks changes randomly every **N** ticks. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for **plot1**, the color is set using the compiler directive #property, and then in the OnCalculate() function the color is set randomly from an earlier prepared list.

```
//+------------------------------------------------------------------+
//|                                                 DRAW_CANDLES.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property description "An indicator to demonstrate DRAW_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate windo
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are
#property description "randomly every N ticks"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- plot Bars
#property indicator_label1  "DRAW_CANDLES1"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- input parameters
input int      N=5;              // The number of ticks to change the type
input int      bars=500;         // The number of bars to show
input bool     messages=false;   // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double         Candle1Buffer1[];
double         Candle1Buffer2[];
double         Candle1Buffer3[];
double         Candle1Buffer4[];
//--- Symbol name
string symbol;
//--- An array to store colors
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- If bars is very small - complete the work ahead of time
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,Candle1Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Candle1Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Candle1Buffer3,INDICATOR_DATA);
   SetIndexBuffer(3,Candle1Buffer4,INDICATOR_DATA);
//--- An empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- The name of the symbol, for which the bars are drawn
   symbol=_Symbol;
```

```
//--- Set the display of the symbol
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_CANDLES("+symbol+")");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=INT_MAX-100;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Select a new symbol from the Market watch window
      symbol=GetRandomSymbolName();
      //--- Change the form
      ChangeLineAppearance();
      //--- Select a new symbol from the Market watch window
      int tries=0;
      //--- Make 5 attempts to fill in the buffers of plot1 with the prices from symbo
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            Candle1Buffer1,Candle1Buffer2,Candle1Buffer3,Candle1Buffer4)
            && tries<5)
        {
         //--- A counter of calls of the CopyFromSymbolToBuffers() function
         tries++;
        }
      //--- Reset the counter of ticks to zero
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Fills in the specified candlestick                               |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[]
                             )
  {
//--- In the rates[] array, we will copy Open, High, Low and Close
   MqlRates rates[];
//--- The counter of attempts
   int attempts=0;
//--- How much has been copied
   int copied=0;
//--- Make 25 attempts to get a timeseries on the desired symbol
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- If failed to copy a sufficient number of bars
   if(copied!=bars)
     {
      //--- Form a message string
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- Show a message in a comment in the main chart window
      Comment(comm);
      //--- Show the message
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- Set the display of the symbol
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
     }
//--- Initialize buffers with empty values
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
   ArrayInitialize(buff4,0.0);
//--- On each tick copy prices to buffers
   for(int i=0;i<copied;i++)
```

```
      {
       //--- Calculate the appropriate index for the buffers
       int buffer_index=total-copied+i;
       //--- Write the prices to the buffers
       buff1[buffer_index]=rates[i].open;
       buff2[buffer_index]=rates[i].high;
       buff3[buffer_index]=rates[i].low;
       buff4[buffer_index]=rates[i].close;
      }
   return(true);
  }
//+------------------------------------------------------------------+
//| Randomly returns a symbol from the Market Watch                  |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- The number of symbols shown in the Market watch window
   int symbols=SymbolsTotal(true);
//--- The position of a symbol in the list - a random number from 0 to symbols
   int number=MathRand()%symbols;
//--- Return the name of a symbol at the specified position
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| Changes the appearance of bars                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the bar properties
   string comm="";
//--- A block for changing the color of bars
   int number=MathRand(); // Get a random number
//--- The divisor is equal to the size of the colors[] array
   int size=ArraySize(colors);
//--- Get the index to select a new color as the remainder of integer division
   int color_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- Write the color
   comm=comm+"\r\n"+(string)colors[color_index];
//--- Write the symbol name
   comm="\r\n"+symbol+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_LINE

The DRAW_COLOR_LINE value is a colored variant of the DRAW_LINE style; it also draws a line using the values of the indicator buffer. But this style, like all color styles with the word **COLOR** in their title has an additional special indicator buffer that stores the color index (number) from a specially set array of colors. Thus, the color of each line segment can be defined by specifying the color index of the index to draw the line at this bar.

The width, style and colors of lines can be set using the compiler directives and dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

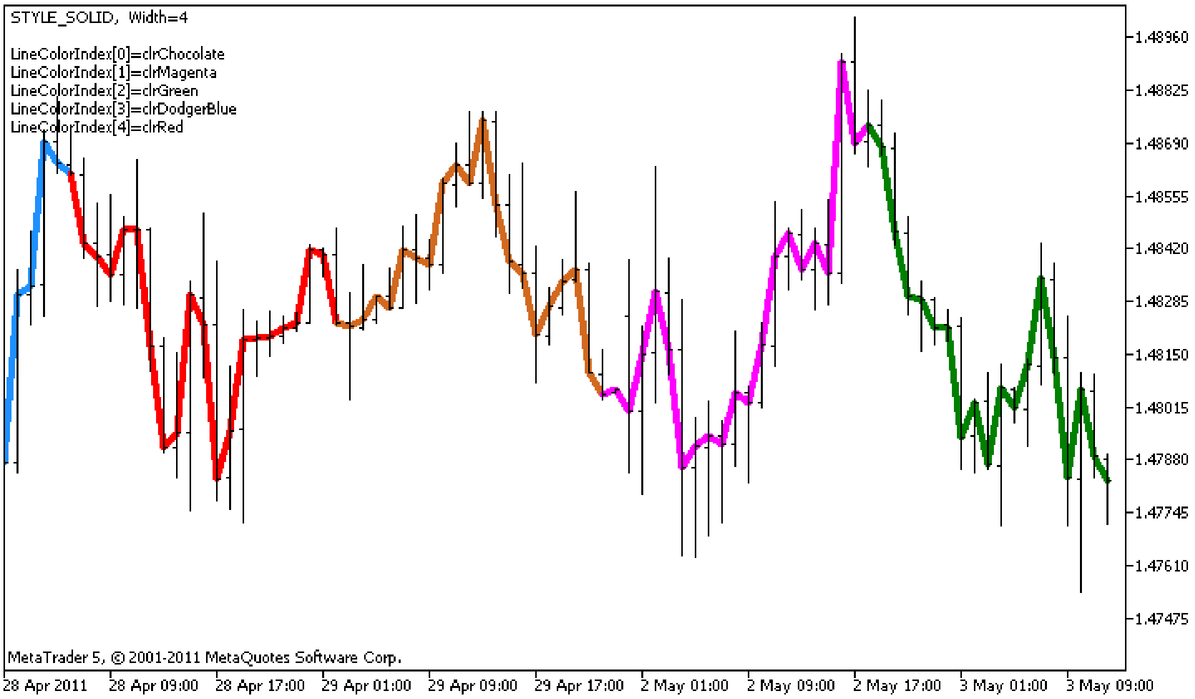
The number of buffers required for plotting DRAW_COLOR_LINE is 2.

- one buffer to store the indicator values used for drawing a line;
- one buffer to store the index of the color of the line on each bar.

Colors can be specified by the compiler directive #property indicator_color1 separated by a comma. The number of colors cannot exceed 64.

```
//--- Define 5 colors for coloring each bar (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 6
```

An example of the indicator that draws a line using Close prices of bars. The line width and style change randomly every N=5 ticks.



The colors of the line segments also change randomly in the custom function ChangeColors().

```
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
```

```
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer devision
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrin
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
```

The example shows the feature of the "color" versions of indicators - to change the color of a line segment, you do not need to change values in the ColorLineColors[] buffer (which contains the color indexes). All you need to do is set new colors in a special array. This allows you to quickly change the color once for the entire plotting, changing only a small array of colors using the PlotIndexSetInteger() function.

Note that initially for **plot1** with DRAW_COLOR_LINE the properties are set using the compiler directive #property, and then in the OnCalculate() function these three properties are set randomly.

The N and Length (the length of color segments in bars) parameters are set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                              DRAW_COLOR_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_LINE"
#property description "It draws a line on Close price in colored pieces of 20 bars eac
#property description "The width, style and color of the line parts are changed rando
```

```
#property description "every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
//--- Define 5 colors for coloring each bar (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (Up to 6
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;           //Number of ticks to change
input int      Length=20;     // The length of each color part in bars
int            line_colors=5; // The number of set colors is 5 - see #property indicat
//--- A buffer for plotting
double         ColorLineBuffer[];
//--- A buffer for storing the line color on each bar
double         ColorLineColors[];

//--- The array for storing colors contains 7 elements
color colors[]={clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGolde
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorLineColors,INDICATOR_COLOR_INDEX);
//--- Initializing the generator of pseudo-random numbers
   MathSrand(GetTickCount());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change the colors of line sections
      ChangeColors(colors,5);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Block for calculating indicator values
   for(int i=0;i<rates_total;i++)
     {
      //--- Write the indicator value into the buffer
      ColorLineBuffer[i]=close[i];
      //--- Now, randomly set a color index for this bar
      int color_index=i%(5*Length);
      color_index=color_index/Length;
      //--- For this bar, the line will have the color with the index color_index
      ColorLineColors[i]=color_index;
     }

//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer devision
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of a displayed line in the indicator      |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_SECTION

The DRAW_COLOR_SECTION style is a color version of DRAW_SECTION, but unlike the latter, it allows drawing sections of different colors. The DRAW_COLOR_SECTION style, like all color styles with the word **COLOR** in their title, contains an additional special indicator buffer that stores the color index (number) from a specially set array of colors. Thus, the color of each section can be defined by specifying the color index of the index of the bar that corresponds to the section end.

The width, color and style of the sections can be specified like for the DRAW_SECTION style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

Sections are drawn from one non-empty value to another non-empty value of the indicator buffer, empty values are ignored. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property: For example, if the indicator should be drawn as a sequence of sections on non-zero values, then you need to set the zero value as an empty one:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_SECTION,PLOT_EMPTY_VALUE,0);
```

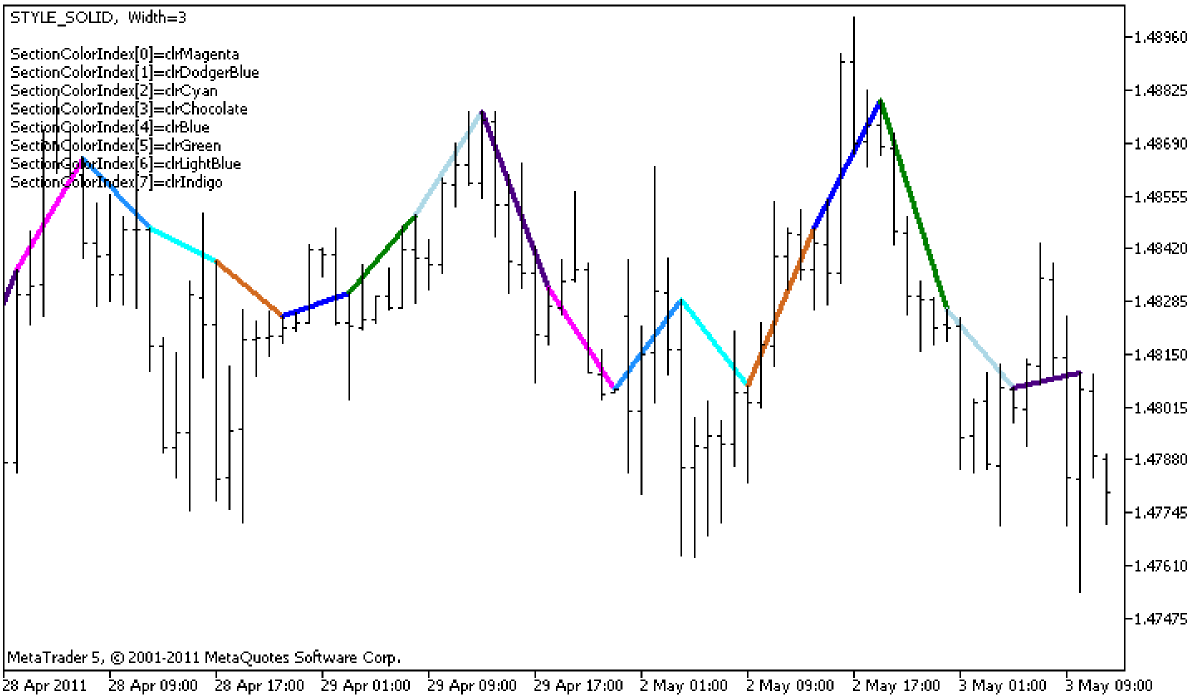
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to the elements that should not be plotted.

The number of buffers required for plotting DRAW_COLOR_SECTION is 2.

- one buffer to store the indicator values used for drawing a line;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

Colors can be specified by the compiler directive #property indicator_color1 separated by a comma. The number of colors cannot exceed 64.

An example of the indicator that draws colored sections each 5 bars long, using the High price values. The color, width and style of sections change randomly every **N** ticks.



Note that initially for **plot1** with DRAW_COLOR_SECTION 8 colors are set using the compiler directive #property. Then in the OnCalculate() function, colors are set randomly from the array of colors colors[].

The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                           DRAW_COLOR_SECTION.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_SECTION"
#property description "It draws colored sections with the length equal to the specifie
#property description "The color, width and style of sections are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorSection
#property indicator_label1  "ColorSection"
#property indicator_type1   DRAW_COLOR_SECTION
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1  clrRed,clrGold,clrMediumBlue,clrLime,clrMagenta,clrBrown,c
#property indicator_style1  STYLE_SOLID
```

```
#property indicator_width1  1
//--- input parameters
input int      N=5;                      // Number of ticks to change
input int      bars_in_section=5;        // The length of sections in bars
//--- An auxiliary variable to calculate ends of sections
int            divider;
int            color_sections;
//--- A buffer for plotting
double         ColorSectionBuffer[];
//--- A buffer for storing the line color on each bar
double         ColorSectionColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorSectionBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorSectionColors,INDICATOR_COLOR_INDEX);
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- The number of colors to color the sections
   int color_sections=8;   //  see A comment to #property indicator_color1
//--- Check the indicator parameter
   if(bars_in_section<=0)
     {
      PrintFormat("Invalid section length=%d",bars_in_section);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=color_sections*bars_in_section;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change colors used to plot the sections
      ChangeColors(colors,color_sections);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- The number of the bar from which the calculation of indicator values starts
   int start=0;
//--- If the indicator has been calculated before, then set start on the previous bar
   if(prev_calculated>0) start=prev_calculated-1;
//--- Here are all the calculations of the indicator values
   for(int i=start;i<rates_total;i++)
     {
      //--- If the bar number is divisible by the section_length, it means this is the
      if(i%bars_in_section==0)
        {
         //--- Set the end of the section at the High price of this bar
         ColorSectionBuffer[i]=high[i];
         //--- A remainder of the division of the bar number by scetion_length*number_
         int rest=i%divider;
         //Get the number of the color =  from 0 to number_of_colors-1
         int color_indext=rest/bars_in_section;
         ColorSectionColors[i]=color_indext;
        }
      //---If the remainder of the division is equal to bars,
      else
        {
         //--- If nothing happened, ignore the bar - set 0
         else ColorSectionBuffer[i]=0;
        }
     }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer devision
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("SectionColorIndex[%d]=%s \r\n",plot_color_ind,ColorToSt
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of a displayed line in the indicator      |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_HISTOGRAM

The DRAW_COLOR_HISTOGRAM style draws a histogram as a sequence of colored columns from zero to a specified value. Values are taken from the indicator buffer. Each column can have its own color from a predefined set of colors.

The width, color and style of the histogram can be specified like for the DRAW_HISTOGRAM style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

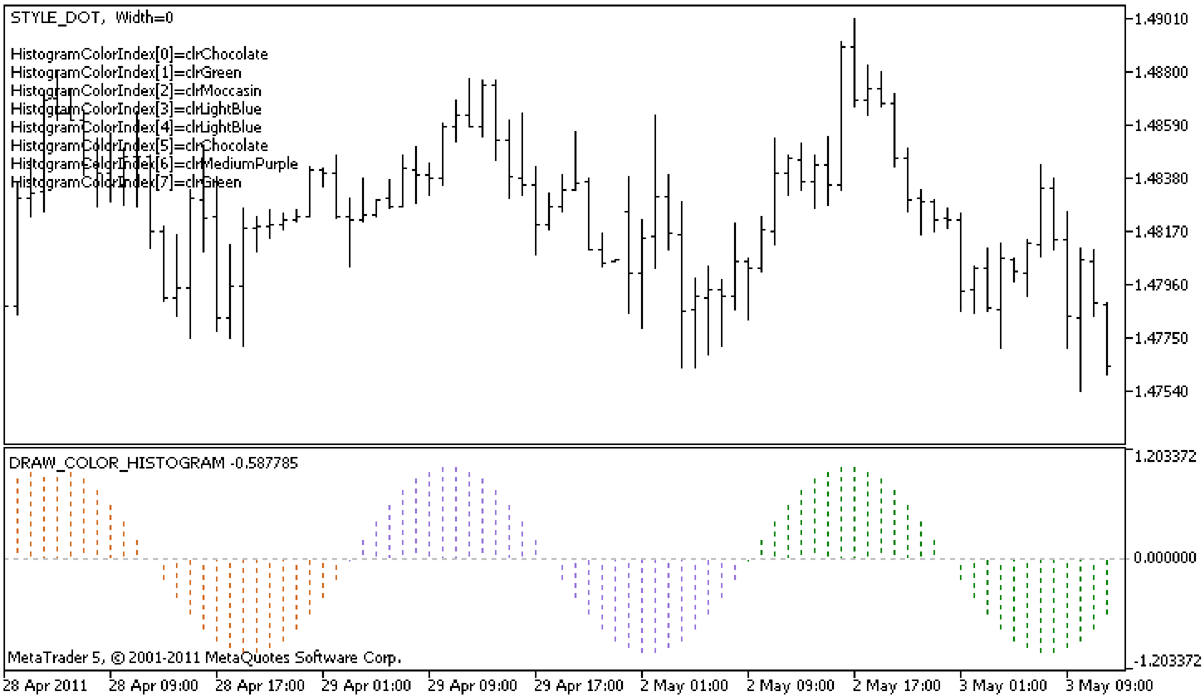
Since a column from the zero level is drawn on each bar, DRAW_COLOR_HISTOGRAM should better be used in a separate chart window. Most often this type of plotting is used to create indicators of the oscillator type, for example, Awesome Oscillator or Market Facilitation Index. For the empty non-displayable values the zero value should be specified.

The number of buffers required for plotting DRAW_COLOR_HISTOGRAM is 2.

- one buffer for storing a non-zero value of the vertical segment on each bar, the second end of the segment is always on the zero line of the indicator;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

Colors can be specified using the compiler directive #property indicator_color1 separated by a comma. The number of colors cannot exceed 64.

An example of the indicator that draws a sinusoid of a specified color based on the MathSin() function. The color, width and style of **all** histogram columns change randomly each **N** ticks. The bars parameter specifies the period of the sinusoid, that is after the specified number of bars the sinusoid will repeat the cycle.


STYLE_DOT, Width=0
HistogramColorIndex[0]=clrChocolate
HistogramColorIndex[1]=clrGreen
HistogramColorIndex[2]=clrMoccasin
HistogramColorIndex[3]=clrLightBlue
HistogramColorIndex[4]=clrLightBlue
HistogramColorIndex[5]=clrChocolate
HistogramColorIndex[6]=clrMediumPurple
HistogramColorIndex[7]=clrGreen
1.49010
1.48800
1.48590
1.48380
1.48170
1.47960
1.47750
1.47540
DRAW_COLOR_HISTOGRAM -0.587785
1.203372
0.000000
-1.203372
MetaTrader 5, © 2001-2011 MetaQuotes Software Corp.
28 Apr 2011
28 Apr 09:00
28 Apr 17:00
29 Apr 01:00
29 Apr 09:00
29 Apr 17:00
2 May 01:00
2 May 09:00
2 May 17:00
3 May 01:00
3 May 09:00

Please note that for **plot1** with the DRAW_COLOR_HISTOGRAM style, 5 colors are set using the compiler directive #property, and then in the OnCalculate() function the colors are selected randomly from the 14 colors stored in the colors[] array. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                       DRAW_COLOR_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM"
#property description "It draws a sinusoid as a histogram in a separate window"
#property description "The color and width of columns are changed randomly"
#property description "after every N ticks"
#property description "The bars parameter sets the number of bars to repeat the sinuso

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- input parameters
input int      bars=30;           // The period of a sinusoid in bars
input int      N=5;               // The number of ticks to change the histogram
//--- plot Color_Histogram
#property indicator_label1  "Color_Histogram"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1  clrRed,clrGreen,clrBlue,clrYellow,clrMagenta,clrCyan,clrMe
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- A buffer of values
double         Color_HistogramBuffer[];
//--- A buffer of color indexes
double         Color_HistogramColors[];
//--- A factor to get the 2Pi angle in radians, when multiplied by the bars parameter
double         multiplier;
int            color_sections;
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Color_HistogramBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,Color_HistogramColors,INDICATOR_COLOR_INDEX);
//---- The number of colors to color the sinusoid
   color_sections=8;   //  see A comment to #property indicator_color1
//--- Calculate the multiplier
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Set the value of bars=%d greater than 1",bars);
      //--- Early termination of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change colors used for the histogram
      ChangeColors(colors,color_sections);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Calculate the indicator values
   int start=0;
```

```
//--- If already calculated during the previous starts of OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculat
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      //--- A value
      Color_HistogramBuffer[i]=sin(i*multiplier);
      //--- Color
      int color_index=i%(bars*color_sections);
      color_index/=bars;
      Color_HistogramColors[i]=color_index;
     }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer devision
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToS
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of a displayed line in the indicator      |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
```

```
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_HISTOGRAM2

The DRAW_COLOR_HISTOGRAM2 style draws a histogram of a specified color – vertical segments using the values of two indicator buffers. But unlike the one-color DRAW_HISTOGRAM2, in this style each column of the histogram can have its own color from a predefined set. The values of all the segments are taken from the indicator buffer.

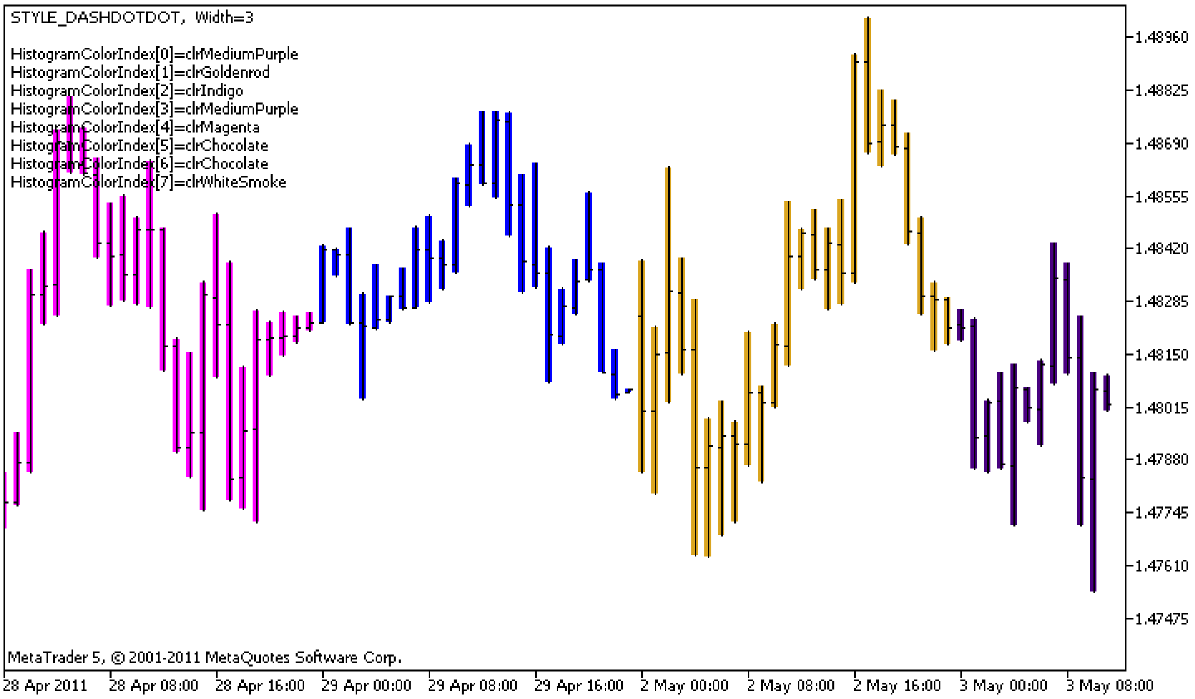
The width, style and color of the histogram can be specified like for theDRAW_HISTOGRAM2 style – using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of the histogram based on the current situation.

The DRAW_COLOR_HISTOGRAM2 style can be used in a separate subwindow of a chart and in its main window. For empty values nothing is drawn, all the values in the indicator buffers need to be set explicitly. Buffers are not initialized with empty values.

The number of buffers required for plotting DRAW_COLOR_HISTOGRAM2 is 3:

- two buffers to store the upper and lower end of the vertical segment on each bar;
- one buffer to store the color index, which is used to draw the segment (it makes sense to set only non-empty values).

An example of the indicator that draws a histogram of a specified color between the High and Low prices. For each day of week, the histogram lines have a different color. The color of the day, width and style of the histogram change randomly each **N** ticks.



Please note that for **plot1** with the DRAW_COLOR_HISTOGRAM2 style, 5 colors are set using the compiler directive #property, and then in the OnCalculate() function the colors are selected randomly from the 14 colors stored in the colors[] array.

The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                        DRAW_COLOR_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_HISTOGRAM2"
#property description "It draws a segment between Open and Close on each bar"
#property description "The color, width and style are changed randomly"
#property description "after every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- plot ColorHistogram_2
#property indicator_label1  "ColorHistogram_2"
#property indicator_type1   DRAW_COLOR_HISTOGRAM2
//--- Define 5 colors for coloring the histogram based on the days of week (they are s
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- input parameter
input int      N=5;              // The number of ticks to change the histogram
int            color_sections;
//--- Value buffers
double         ColorHistogram_2Buffer1[];
double         ColorHistogram_2Buffer2[];
//--- A buffer of color indexes
double         ColorHistogram_2Colors[];
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorHistogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorHistogram_2Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorHistogram_2Colors,INDICATOR_COLOR_INDEX);
//--- Set an empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- The number of colors to color the sinusoid
   color_sections=8;   //  See a comment to #property indicator_color1
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change the colors used to draw the histogram
      ChangeColors(colors,color_sections);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Calculate the indicator values
   int start=0;
//--- To get the day of week by the open price of each bar
   MqlDateTime dt;
//--- If already calculated during the previous starts of OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // set the beginning of the calculat
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
      //--- value
      ColorHistogram_2Buffer1[i]=high[i];
      ColorHistogram_2Buffer2[i]=low[i];
      //--- Set the color index according to the day of week
      int day=dt.day_of_week;
      ColorHistogram_2Colors[i]=day;
     }
//--- Return the prev_calculated value for the next call of the function
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer division
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorTo
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of a displayed line in the indicator      |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
```

```
   comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm=EnumToString(styles[style_index])+", "+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_ARROW

The DRAW_COLOR_ARROW style draws colored arrows (symbols of the set Wingdings) based on the value of the indicator buffer. In contrast to DRAW_ARROW, in this style it is possible to set a color from a predefined set of colors specified by the indicator_color1 property for each symbol.

The width and color of the symbols can be specified like for theDRAW_ARROW style – using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows changing the look of an indicator based on the current situation.

The symbol code is set using the PLOT_ARROW property.

```
//--- Define the symbol code from the Wingdings font to draw in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

The default value of PLOT_ARROW=159 (a circle).

Each arrow is actually a symbol that has the height and the anchor point, and can cover some important information on a chart (for example, the closing price at the bar). Therefore, we can additionally specify the vertical shift in pixels, which does not depend on the scale of the chart. The arrows will be shifted down by the specified number of pixels, although the values of the indicator will remain the same:

```
//--- Set the vertical shift of arrows in pixels
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

A negative value of PLOT_ARROW_SHIFT means the shift of arrows upwards, a positive values shifts the arrow down.

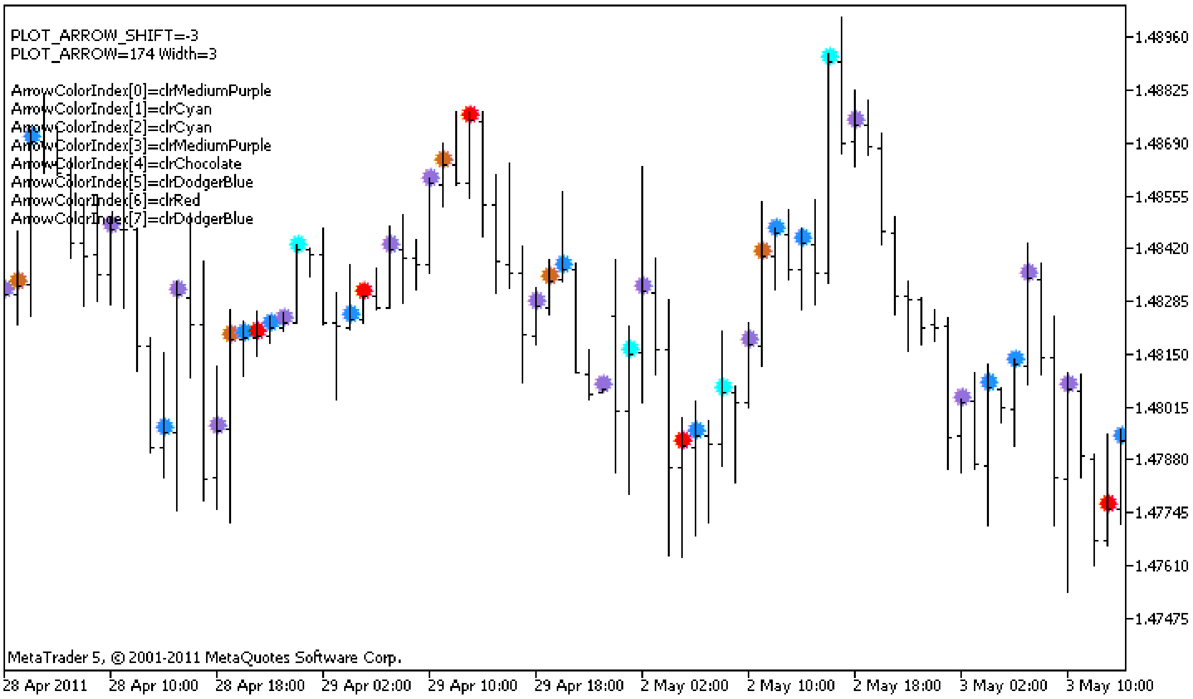
The DRAW_COLOR_ARROW style can be used in a separate subwindow of a chart and in its main window. Empty values are not drawn and do not appear in the "Data Window", all the values in the indicator buffers should be set explicitly. Buffers are not initialized with a zero value.

```
//--- Set an empty value
   PlotIndexSetDouble(DRAW_COLOR_ARROW_plot_index,PLOT_EMPTY_VALUE,0);
```

The number of buffers required for plotting DRAW_COLOR_ARROW is 2.

- a buffer to store the value of the price which is used to draw the symbol (plus a shift in pixels, given in the PLOT_ARROW_SHIFT property);
- a buffer to store the color index, which is used to draw an arrow (it makes sense to set only non-empty values).

An example of the indicator, which draws arrows on each bar with the close price higher than the close price of the previous bar. The width, shift and symbol code of **all** arrows are changed randomly every **N** ticks. The color of the symbol depends on the number of the bar on which it is drawn.



PLOT_ARROW_SHIFT=-3
PLOT_ARROW=174 Width=3

ArrowColorIndex[0]=clrMediumPurple
ArrowColorIndex[1]=clrCyan
ArrowColorIndex[2]=clrCyan
ArrowColorIndex[3]=clrMediumPurple
ArrowColorIndex[4]=clrChocolate
ArrowColorIndex[5]=clrDodgerBlue
ArrowColorIndex[6]=clrRed
ArrowColorIndex[7]=clrDodgerBlue

In the example, for **plot1** with the DRAW_COLOR_ARROW style, the properties, color and size are specified using the compiler directive #property, and then in the OnCalculate() function the properties are set randomly. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

Please note that initially 8 colors are set using the compiler directive #property, and then in the OnCalculate() function, the color is set randomly from the 14 colors that are stored in the colors[] array.

```
//+------------------------------------------------------------------+
//|                                          DRAW_COLOR_ARROW.mq5 |
//|                      Copyright 2011, MetaQuotes Software Corp. |
//|                                           https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ARROW"
#property description "Draws different-color arrows set by Unicode characters, on a ch
#property description "The color, size, shift and symbol code of the arrow are changed
#property description " randomly every N ticks"
#property description "The code parameter sets the base value: code=159 (a circle)"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorArrow
#property indicator_label1  "ColorArrow"
#property indicator_type1   DRAW_COLOR_ARROW
```

```
//--- Define 8 colors for coloring the histogram (they are stored in the special array
#property indicator_color1  clrRed,clrBlue,clrSeaGreen,clrGold,clrDarkOrange,clrMagent
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- input parameters
input int      N=5;         // Number of ticks to change
input ushort   code=159;    // Symbol code to draw in DRAW_ARROW
int            color_sections;
//--- An indicator buffer for the plot
double         ColorArrowBuffer[];
//--- A buffer to store color indexes
double         ColorArrowColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorArrowBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorArrowColors,INDICATOR_COLOR_INDEX);
//--- Define the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- Set the vertical shift of arrows in pixels
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- Set as an empty value 0
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- The number of colors to color the sinusoid
   color_sections=8;   //  see a comment to #property indicator_color1
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the color, size, shift and code of the arrow
   ticks++;
//--- If a critical number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change arrow properties
      ChangeLineAppearance();
      //--- Change the colors used to draw the histogram
      ChangeColors(colors,color_sections);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- Block for calculating indicator values
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- Calculation loop
   for(int i=1;i<rates_total;i++)
     {
      //--- If the current Close price is higher than the previous one, draw an arrow
      if(close[i]>close[i-1])
         ColorArrowBuffer[i]=close[i];
      //--- Otherwise specify the null value
      else
         ColorArrowBuffer[i]=0;
      //--- Arrow color
      int index=i%color_sections;
      ColorArrowColors[i]=index;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of line segments                               |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer division
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("ArrowColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrin
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of a displayed line in the indicator      |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the line properties
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5; // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+" Width="+IntegerToString(width);

//--- A block for changing the arrow code (PLOT_ARROW)
   number=MathRand();
//--- Get the remainder of integer division to calculate a new code of the arrow (fro
   int code_add=number%20;
//--- Set the new symbol code as the result of code+code_add
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- Write the symbol code PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- A block for changing the vertical shift of arrows in pixels
   number=MathRand();
//--- Get the shift as the remainder of the integer division
   int shift=20-number%41;
//--- Set the new shift from
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- Write the shift PLOT_ARROW_SHIFT
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_ZIGZAG

The DRAW_COLOR_ZIGZAG style draws segments of different colors, using the values of two indicator buffers. This style is a colored version of DRAW_ZIGZAG, i.e. allows specifying for each segment an individual color from the predefined set of colors. The segments are plotted from a value in the first buffer to a value in the second indicator buffer. None of the buffers can contain only empty values, since in this case nothing is plotted.

The width, color and style of the line can be specified like for the DRAW_ZIGZAG style - using compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

Sections are drawn from a non-empty value of one buffer to a non-empty value of another indicator buffer. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

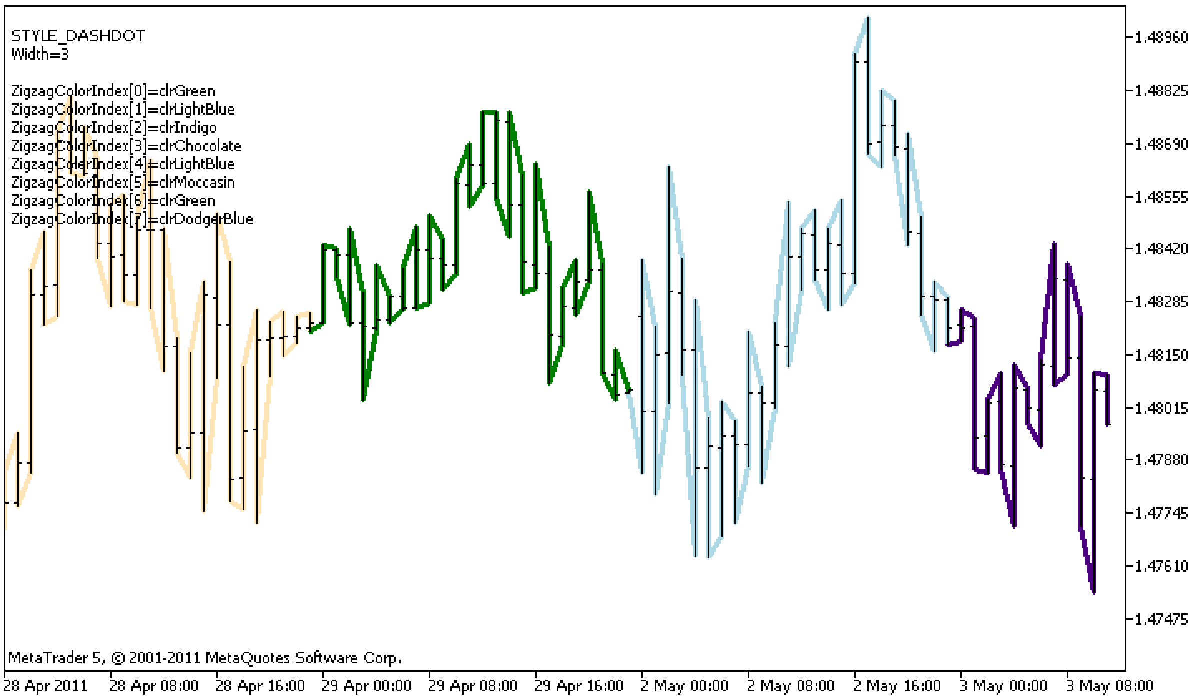
```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

Always explicitly fill in the of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_COLOR_ZIGZAG is 3:

- two buffers to store the values of ends of the zigzag sections;
- one buffer to store the color index, which is used to draw the section (it makes sense to set only non-empty values).

An example of the indicator that plots a saw based on the High and Low prices. The color, width and style of the zigzag lines change randomly every **N** ticks.



Please note that for **plot1** with the DRAW_COLOR_ZIGZAG style, 8 colors are set using the compiler directive #property, and then in the OnCalculate() function the color is selected randomly from the 14 colors stored in the colors[] array.

The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).

```
//+------------------------------------------------------------------+
//|                                            DRAW_COLOR_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_ZIGZAG"
#property description "It draws a broken line as a sequence of colored sections, the c
#property description "The color, width and style of segments are changed randomly"
#property description " every N ticks"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- plot Color_Zigzag
#property indicator_label1  "Color_Zigzag"
#property indicator_type1   DRAW_COLOR_ZIGZAG
//--- Define 8 colors for coloring sections (they are stored in a special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameter
input int      N=5;              // Number of ticks to change
int            color_sections;
//--- Buffers of values of segment ends
double         Color_ZigzagBuffer1[];
double         Color_ZigzagBuffer2[];
//--- Buffers of color indexes of segment ends
double         Color_ZigzagColors[];
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//--- An array to store the line styles
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Color_ZigzagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Color_ZigzagBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Color_ZigzagColors,INDICATOR_COLOR_INDEX);
//----Number of color for coloring the zigzag
   color_sections=8;   //  see a comment to the #property indicator_color1 property
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Calculate ticks to change the style, color and width of the line
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change colors used to plot the sections
      ChangeColors(colors,color_sections);
      //--- Reset the counter of ticks to zero
      ticks=0;
     }

//--- The structure of time is required to get the day of week of each bar
   MqlDateTime dt;

//--- The start position of calculations
   int start=0;
//--- If the indicator was calculated on the previous tick, then start the calculatio
   if(prev_calculated!=0) start=prev_calculated-1;
//--- Calculation loop
   for(int i=start;i<rates_total;i++)
```

```
     {
      //--- Write the bar open time in the structure
      TimeToStruct(time[i],dt);

      //--- If the bar number is even
      if(i%2==0)
        {
         //---  Write High in the 1st buffer and Low in the 2nd one
         Color_ZigzagBuffer1[i]=high[i];
         Color_ZigzagBuffer2[i]=low[i];
         //--- The color of the segment
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
      //--- the bar number is odd
      else
        {
         //--- Fill in the bar in a reverse order
         Color_ZigzagBuffer1[i]=low[i];
         Color_ZigzagBuffer2[i]=high[i];
         //--- The color of the segment
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Changes the color of the zigzag segments                         |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer division
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("ZigzagColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of the zigzag segments                    |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the properties of Color_ZigZag
   string comm="";
//--- A block for changing the width of the line
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- A block for changing the style of the line
   number=MathRand();
//--- The divisor is equal to the size of the styles array
   int size=ArraySize(styles);
//--- Get the index to select a new style as the remainder of integer division
   int style_index=number%size;
//--- Set the color as the PLOT_LINE_COLOR property
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- Write the line style
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_BARS

The DRAW_COLOR_BARS style draws bars on the values of four indicator buffers, which contain the Open, High, Low and Close prices. This style is an advanced version of DRAW_BARS and allows specifying for each bar an individual color from the predefined set of colors. It used for creating custom indicators as bars, including those in a separate subwindow of a chart and on other financial instruments.

The color of bars can be set using the compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of **all** four indicator buffers are set. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_BARS,PLOT_EMPTY_VALUE,0);
```

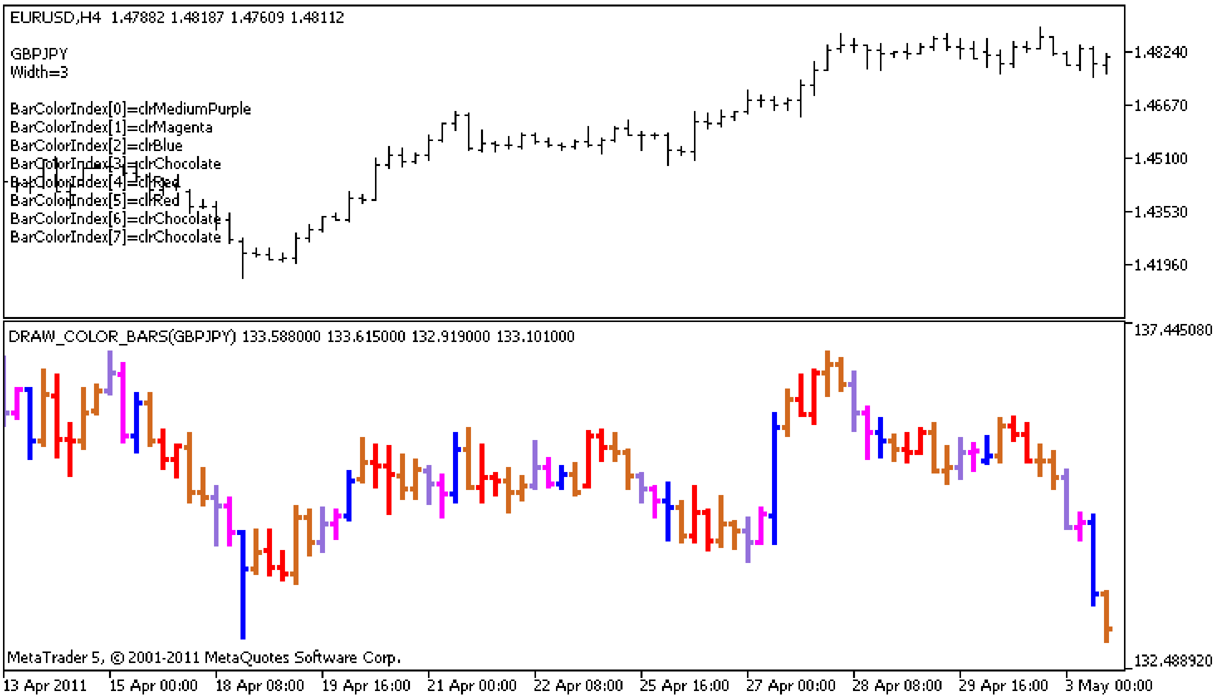
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of buffers required for plotting DRAW_COLOR_BARS is 5:

- four buffer to store the values of Open, High, Low and Close;
- one buffer to store the color index, which is used to draw a bar (it makes sense to set it only for the bars that will be drawn).

All buffers for the plotting should go one after the other in the given order: Open, High, Low, Close and the color buffer. None of the price buffers can contain only null values, since in this case nothing is plotted.

An example of the indicator that draws bars on a selected financial instrument in a separate window. The color of bars changes randomly every **N** ticks. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).


EURUSD,H4 1.47882 1.48187 1.47609 1.48112

GBPJPY
Width=3

BarColorIndex[0]=clrMediumPurple
BarColorIndex[1]=clrMagenta
BarColorIndex[2]=clrBlue
BarColorIndex[3]=clrChocolate
BarColorIndex[4]=clrRed
BarColorIndex[5]=clrRed
BarColorIndex[6]=clrChocolate
BarColorIndex[7]=clrChocolate

DRAW_COLOR_BARS(GBPJPY) 133.588000 133.615000 132.919000 133.101000

1.48240
1.46670
1.45100
1.43530
1.41960
137.445080
132.488920
MetaTrader 5, © 2001-2011 MetaQuotes Software Corp.
13 Apr 2011
15 Apr 00:00
18 Apr 08:00
19 Apr 16:00
21 Apr 00:00
22 Apr 08:00
25 Apr 16:00
27 Apr 00:00
28 Apr 08:00
29 Apr 16:00
3 May 00:00

Please note that for **plot1** with the DRAW_COLOR_BARS style, 8 colors are set using the compiler directive #property, and then in the OnCalculate() function the color is selected randomly from the 14 colors stored in the colors[] array.

```
//+------------------------------------------------------------------+
//|                                              DRAW_COLOR_BARS.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_BARS"
#property description "It draws different-color bars of a selected symbol in a separat
#property description "The color and width of bars, as well as the symbol are changed
#property description "every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot ColorBars
#property indicator_label1  "ColorBars"
#property indicator_type1   DRAW_COLOR_BARS
//--- Define 8 colors for coloring bars (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrLi
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input   int    N=5;              // The number of ticks to change the type
input int      bars=500;         // The number of bars to show
input bool     messages=false;   // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double         ColorBarsBuffer1[];
double         ColorBarsBuffer2[];
double         ColorBarsBuffer3[];
double         ColorBarsBuffer4[];
double         ColorBarsColors[];
//--- Symbol name
string symbol;
int    bars_colors;
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorBarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBarsBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorBarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorBarsBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorBarsColors,INDICATOR_COLOR_INDEX);
//---- Number of colors for coloring bars
   bars_colors=8;   //  see a comment to the #property indicator_color1 property
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=0;
//--- Count ticks to change the style, color and width of the bar
```

```
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Select a new symbol from the Market watch window
      symbol=GetRandomSymbolName();
      //--- Change the line properties
      ChangeLineAppearance();
      //--- Change the colors used to draw the candlesticks
      ChangeColors(colors,bars_colors);
      int tries=0;
      //--- Make 5 attempts to fill in the buffers with the prices from symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total,bars_colors) && tries<5)
        {
         //--- A counter of calls of the CopyFromSymbolToBuffers() function
         tries++;
        }
      //--- Reset the counter of ticks to zero
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Fill in the indicator buffers with prices                        |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total,int bar_colors)
  {
//--- In the rates[] array, we will copy Open, High, Low and Close
   MqlRates rates[];
//--- The counter of attempts
   int attempts=0;
//--- How much has been copied
   int copied=0;
//--- Make 25 attempts to get a timeseries on the desired symbol
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- If failed to copy a sufficient number of bars
   if(copied!=bars)
     {
      //--- Form a message string
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- Show a message in a comment in the main chart window
      Comment(comm);
      //--- Show the message
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- Set the display of the symbol
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_BARS("+name+")");
     }
//--- Initialize buffers with empty values
   ArrayInitialize(ColorBarsBuffer1,0.0);
   ArrayInitialize(ColorBarsBuffer2,0.0);
   ArrayInitialize(ColorBarsBuffer3,0.0);
   ArrayInitialize(ColorBarsBuffer4,0.0);

//--- Copy prices to the buffers
   for(int i=0;i<copied;i++)
     {
      //--- Calculate the appropriate index for the buffers
      int buffer_index=total-copied+i;
      //--- Write the prices to the buffers
      ColorBarsBuffer1[buffer_index]=rates[i].open;
      ColorBarsBuffer2[buffer_index]=rates[i].high;
      ColorBarsBuffer3[buffer_index]=rates[i].low;
      ColorBarsBuffer4[buffer_index]=rates[i].close;
      //---
      ColorBarsColors[buffer_index]=i%bar_colors;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| Randomly returns a symbol from the Market Watch                  |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- The number of symbols shown in the Market watch window
   int symbols=SymbolsTotal(true);
//--- The position of a symbol in the list - a random number from 0 to symbols
   int number=MathRand()%symbols;
//--- Return the name of a symbol at the specified position
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| Changes the color of the zigzag segments                         |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer division
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("BarColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of bars                                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the bar properties
   string comm="";

//--- A block for changing the width of bars
   int number=MathRand();
//--- Get the width of the remainder of integer division
   int width=number%5;   // The width is set from 0 to 4
//--- Set the color as the PLOT_LINE_WIDTH property
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- Write the line width
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- Write the symbol name
   comm="\r\n"+symbol+comm;

//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

DRAW_COLOR_CANDLES

The DRAW_COLOR_CANDLES style, like DRAW_CANDLES, draws candlesticks using the values of four indicator buffers, which contain Open, High, Low and Close prices. In addition, it allows specifying a color for each candlestick from a given set. For this purpose, the style has a special color buffer that stores color indexes for each bar. It used for creating custom indicators as a sequence of candlesticks, including those in a separate subwindow of a chart and on other financial instruments.

The number of colors of candlesticks can be set using the compiler directives or dynamically using the PlotIndexSetInteger() function. Dynamic changes of the plotting properties allows "to enliven" indicators, so that their appearance changes depending on the current situation.

The indicator is drawn only to those bars, for which non-empty values of four price buffers of the indicator are set. To specify what value should be considered as "empty", set this value in the PLOT_EMPTY_VALUE property:

```
//--- The 0 (empty) value will mot participate in drawing
   PlotIndexSetDouble(index_of_plot_DRAW_COLOR_CANDLES,PLOT_EMPTY_VALUE,0);
```

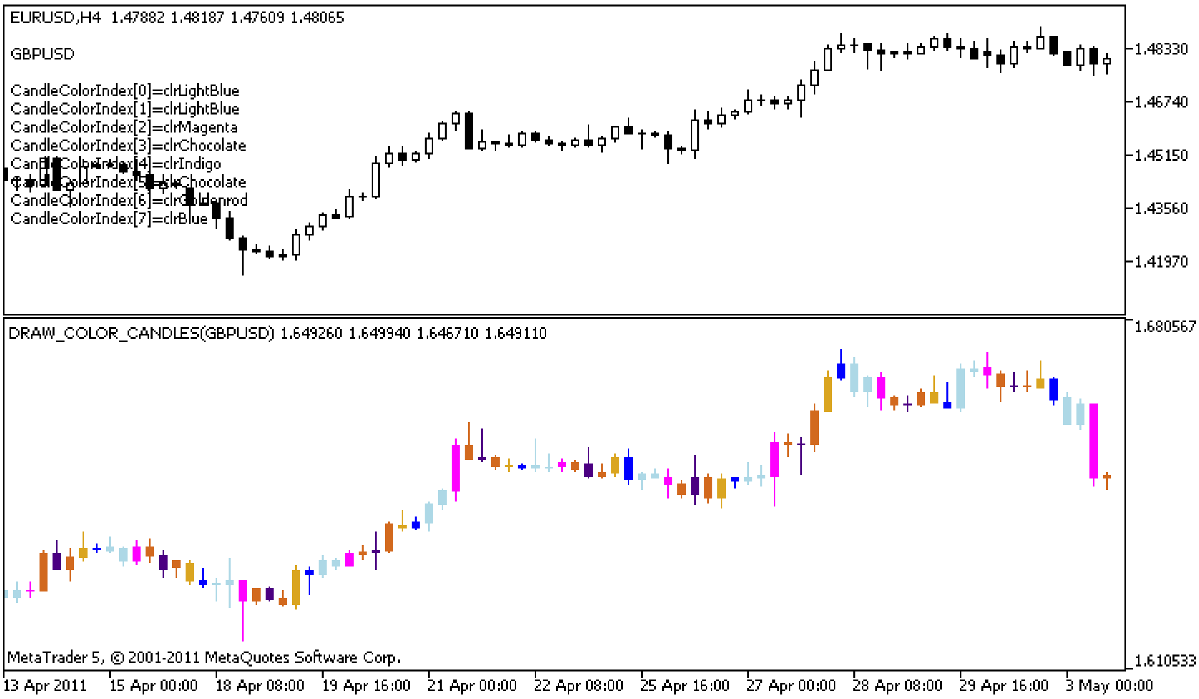
Always explicitly fill in the values of the indicator buffers, set an empty value in a buffer to skip bars.

The number of required buffers for plotting DRAW_COLOR_CANDLES is 5:

- four buffer to store the values of Open, High, Low and Close;
- one buffer to store the color index, which is used to draw a candlestick (it makes sense to set it only for the candlesticks that will be drawn).

All buffers for the plotting should go one after the other in the given order: Open, High, Low, Close and the color buffer. None of the price buffers can contain only empty values, since in this case nothing is plotted.

An example of the indicator that draws candlesticks for a selected financial instrument in a separate window. The color of candlesticks changes randomly every **N** ticks. The N parameter is set in external parameters of the indicator for the possibility of manual configuration (the Parameters tab in the indicator's Properties window).



Please note that for **plot1**, the color is set using the compiler directive #property, and then in the OnCalculate() function the color is set randomly from an earlier prepared list.

```
//+------------------------------------------------------------------+
//|                                           DRAW_COLOR_CANDLES.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An indicator to demonstrate DRAW_COLOR_CANDLES."
#property description "It draws candlesticks of a selected symbol in a separate windo
#property description " "
#property description "The color and width of candlesticks, as well as the symbol are
#property description "randomly every N ticks"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
//--- Define 8 colors for coloring candlesticks (they are stored in the special array)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```
//--- input parameters
input int      N=5;              // The number of ticks to change the type
input int      bars=500;         // The number of candlesticks to show
input bool     messages=false;   // Show messages in the "Expert Advisors" log
//--- Indicator buffers
double         ColorCandlesBuffer1[];
double         ColorCandlesBuffer2[];
double         ColorCandlesBuffer3[];
double         ColorCandlesBuffer4[];
double         ColorCandlesColors[];
int            candles_colors;
//--- Symbol name
string symbol;
//--- An array for storing colors contains 14 elements
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- If bars is very small - complete the work ahead of time
   if(bars<50)
     {
      Comment("Please specify a larger number of bars! The operation of the indicator
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorCandlesBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorCandlesBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorCandlesBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorCandlesBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- An empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- The name of the symbol, for which the bars are drawn
   symbol=_Symbol;
//--- Set the display of the symbol
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symb
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
//---- The number of colors to color candlesticks
   candles_colors=8;      //  see. a comment to the #property indicator_color1 propert
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int ticks=INT_MAX-100;
//--- Count ticks to change the style and color
   ticks++;
//--- If a sufficient number of ticks has been accumulated
   if(ticks>=N)
     {
      //--- Select a new symbol from the Market watch window
      symbol=GetRandomSymbolName();
      //--- Change the form
      ChangeLineAppearance();
      //--- Change the colors used to draw the candlesticks
      ChangeColors(colors,candles_colors);

      int tries=0;
      //--- Make 5 attempts to fill in the buffers of plot1 with the prices from symbo
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            ColorCandlesBuffer1,ColorCandlesBuffer2,ColorCandlesBuffer3,
            ColorCandlesBuffer4,ColorCandlesColors,candles_colors)
            && tries<5)
        {
         //--- A counter of calls of the CopyFromSymbolToBuffers() function
         tries++;
        }
      //--- Reset the counter of ticks to zero
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Fills in the specified candlestick                               |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[],
                             double &col_buffer[],
                             int    cndl_colors
                             )
  {
//--- In the rates[] array, we will copy Open, High, Low and Close
   MqlRates rates[];
//--- The counter of attempts
   int attempts=0;
//--- How much has been copied
   int copied=0;
//--- Make 25 attempts to get a timeseries on the desired symbol
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attemp
     }
//--- If failed to copy a sufficient number of bars
   if(copied!=bars)
     {
      //--- Form a message string
      string comm=StringFormat("For the symbol %s, managed to receive only %d bars of
                               name,
                               copied,
                               bars
                               );
      //--- Show a message in a comment in the main chart window
      Comment(comm);
      //--- Show the message
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- Set the display of the symbol
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
     }
//--- Initialize buffers with empty values
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
   ArrayInitialize(buff4,0.0);
//--- On each tick copy prices to buffers
   for(int i=0;i<copied;i++)
     {
      //--- Calculate the appropriate index for the buffers
      int buffer_index=total-copied+i;
      //--- Write the prices to the buffers
      buff1[buffer_index]=rates[i].open;
      buff2[buffer_index]=rates[i].high;
      buff3[buffer_index]=rates[i].low;
      buff4[buffer_index]=rates[i].close;
      //--- Set the candlestick color
      int color_index=i%cndl_colors;
      col_buffer[buffer_index]=color_index;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| Randomly returns a symbol from the Market Watch                  |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- The number of symbols shown in the Market watch window
   int symbols=SymbolsTotal(true);
//--- The position of a symbol in the list - a random number from 0 to symbols
   int number=MathRand()%symbols;
//--- Return the name of a symbol at the specified position
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| Changes the color of the candlestick segments                    |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- The number of colors
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- For each color index define a new color randomly
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- Get a random value
      int number=MathRand();
      //--- Get an index in the col[] array as a remainder of the integer division
      int i=number%size;
      //--- Set the color for each index as the property PLOT_LINE_COLOR
      PlotIndexSetInteger(0,                    //  The number of a graphical style
                          PLOT_LINE_COLOR,      //  Property identifier
                          plot_color_ind,       //  The index of the color, where we w
                          cols[i]);             //  A new color
      //--- Write the colors
      comm=comm+StringFormat("CandleColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| Changes the appearance of candlesticks                           |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- A string for the formation of information about the candlestick properties
   string comm="";
//--- Write the symbol name
   comm="\r\n"+symbol+comm;
//--- Show the information on the chart using a comment
   Comment(comm);
  }
```

Connection between Indicator Properties and Corresponding Functions

Every custom indicator has numerous properties, some of which are obligatory and are always positioned at the beginning of description. They are the following properties:

- indication of a window to plot the indicator – indicator_separate_window or indicator_chart_window;
- number of indicator buffers – indicator_buffers;
- number of plots of the indicator – indicator_plots.

Also there are other properties that can be set both through preprocessor directives and through functions intended for custom indicator creation. These properties and corresponding functions are described in the following table.

Directives for properties of indicator subwindow	Functions of IndicatorSet...() type	Description of the adjusted property of the subwindow
indicator_height	IndicatorSetInteger (INDICATOR_INDICATOR_HEIGHT, nHeight)	The fixed value of the subwindow height
indicator_minimum	IndicatorSetDouble (INDICATOR_MINIMUM, dMaxValue)	Minimal value of the vertical axis
indicator_maximum	IndicatorSetDouble (INDICATOR_MAXIMUM, dMinValue)	Maximal value of the vertical axis
indicator_levelN	IndicatorSetDouble (INDICATOR_LEVELVALUE, N-1, nLevelValue)	Vertical axis value for N level
no preprocessor directive	IndicatorSetString (INDICATOR_LEVELTEXT, N-1, sLevelName)	Name of a displayed level
indicator_levelcolor	IndicatorSetInteger (INDICATOR_LEVELCOLOR, N-1, nLevelColor)	Color of N level
indicator_levelwidth	IndicatorSetInteger (INDICATOR_LEVELWIDTH, N-1, nLevelWidth)	Line width for N level
indicator_levelstyle	IndicatorSetInteger (INDICATOR_LEVELSTYLE, N-1, nLevelStyle)	Line style for N level
Directives for plotting properties	**Functions of PlotIndexSet...() type**	**Description of the adjusted property of a plot**
indicator_labelN	PlotIndexSetString(N-1,PLOT_LABEL,sLabel)	Short name of the number N plot. It is displayed in

Directives for properties of indicator subwindow	Functions of IndicatorSet...() type	Description of the adjusted property of the subwindow
		DataWindow and in the pop-up tooltip when pointing the mouse cursor over it
indicator_colorN	PlotIndexSetInteger(N-1, PLOT_LINE_COLOR, nColor)	Line color for N plot
indicator_styleN	PlotIndexSetInteger(N-1, PLOT_LINE_STYLE, nType)	Line style for N plot
indicator_typeN	PlotIndexSetInteger(N-1, PLOT_DRAW_TYPE, nType)	Line type for N plot
indicator_widthN	PlotIndexSetInteger(N-1, PLOT_LINE_WIDTH, nWidth)	Line width for N plot
Common indicator properties	**Functions of IndicatorSet...() type**	**Description**
no preprocessor directive	IndicatorSetString (INDICATOR_SHORTNAME, sShortName)	Sets the convenient short name of the indicator that will be displayed in the list of indicators (opened in the terminal by pressing **Ctrl+I**).
no preprocessor directive	IndicatorSetInteger (INDICATOR_DIGITS, nDigits)	Sets required accuracy of display of indicator values - number of decimal places
no preprocessor directive	IndicatorSetInteger (INDICATOR_LEVELS, nLevels)	Sets number of levels in the indicator window
indicator_applied_price	No function, the property can be set only by the preprocessor directive.	Default price type used for the indicator calculation. It is specified when necessary, only if OnCalculate() of the first type is used. The property value can also be set from the dialog of indicator properties in the "Parameters" tab - "Apply to".

It should be noted that numeration of levels and plots in preprocessor terms starts with one, while numeration of the same properties at using functions starts with zero, i.e. the indicated value must be by 1 less than that indicated for #property.

There are several directives, for which there are no corresponding functions:

Directive	Description
indicator_chart_window	Indicator is displayed in the main window

Directive	Description
indicator_separate_window	Indicator is displayed in a separate subwindow
indicator_buffers	Number of required indicator buffers
indicator_plots	Number of plots in the indicator

SetIndexBuffer

The function binds a specified indicator buffer with one-dimensional dynamic array of the double type.

```
bool  SetIndexBuffer(
   int                    index,         // buffer index
   double                 buffer[],      // array
   ENUM_INDEXBUFFER_TYPE  data_type      // what will be stored
   );
```

Parameters

index

[in] Number of the indicator buffer. The numbering starts with 0. The number must be less than the value declared in #property indicator_buffers.

buffer[]

[in] An array declared in the custom indicator program.

data_type

[in] Type of data stored in the indicator array. By default it is INDICATOR_DATA (values of the calculated indicator). It may also take the value of INDICATOR_COLOR_INDEX; in this case this buffer is used for storing color indexes for the previous indicator buffer. You can specify up to 64 colors in the #property indicator_colorN line. The INDICATOR_CALCULATIONS value means that the buffer is used in intermediate calculations of the indicator and is not intended for drawing.

Return Value

If successful, returns true, otherwise - false.

Note

After binding, the dynamic array buffer[] will be indexed as in common arrays, even if the indexing of timeseries is pre-installed for the bound array. If you want to change the order of access to elements of the indicator array, use the ArraySetAsSeries() function **after binding the array using the SetIndexBuffer() function**. Please note that you can't change the size for dynamic arrays set as indicator buffers by the function SetIndexBuffer(). For indicator buffers, all operations of size changes are performed by the executing sub-system of the terminal.

Example:

```
//+------------------------------------------------------------------+
//|                                              TestCopyBuffer1.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
```

```
//---- plot MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool                 AsSeries=true;
input int                  period=15;
input ENUM_MA_METHOD       smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int                  shift=0;
//--- indicator buffers
double                     MABuffer[];
int                        ma_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   if(AsSeries) ArraySetAsSeries(MABuffer,true);
   Print("Indicator buffer is timeseries = ",ArrayGetAsSeries(MABuffer));
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Indicator buffer after SetIndexBuffer() is timeseries = ",
         ArrayGetAsSeries(MABuffer));

//--- change the order of accessing elements of the indicator buffer
   ArraySetAsSeries(MABuffer,AsSeries);

   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
```

```
//--- Copy the values of the moving average in the buffer MABuffer
   int copied=CopyBuffer(ma_handle,0,0,rates_total,MABuffer);

   Print("MABuffer[0] = ",MABuffer[0]);// Depending on the value AsSeries
                                       // Will receive a very old value
                                       // Or for the current unfinished bar

//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

See also

[Custom Indicator Properties](), [Access to timeseries and indicators]()

IndicatorSetDouble

The function sets the value of the corresponding indicator property. Indicator property must be of the double type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetDouble(
   int      prop_id,           // identifier
   double   prop_value         // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetDouble(
   int      prop_id,           // identifier
   int      prop_modifier,     // modifier
   double   prop_value         // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_DOUBLE enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier. Numbering of levels starts from 0. It means that in order to set property for the second level you need to specify 1 (1 less than when using compiler directive).

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

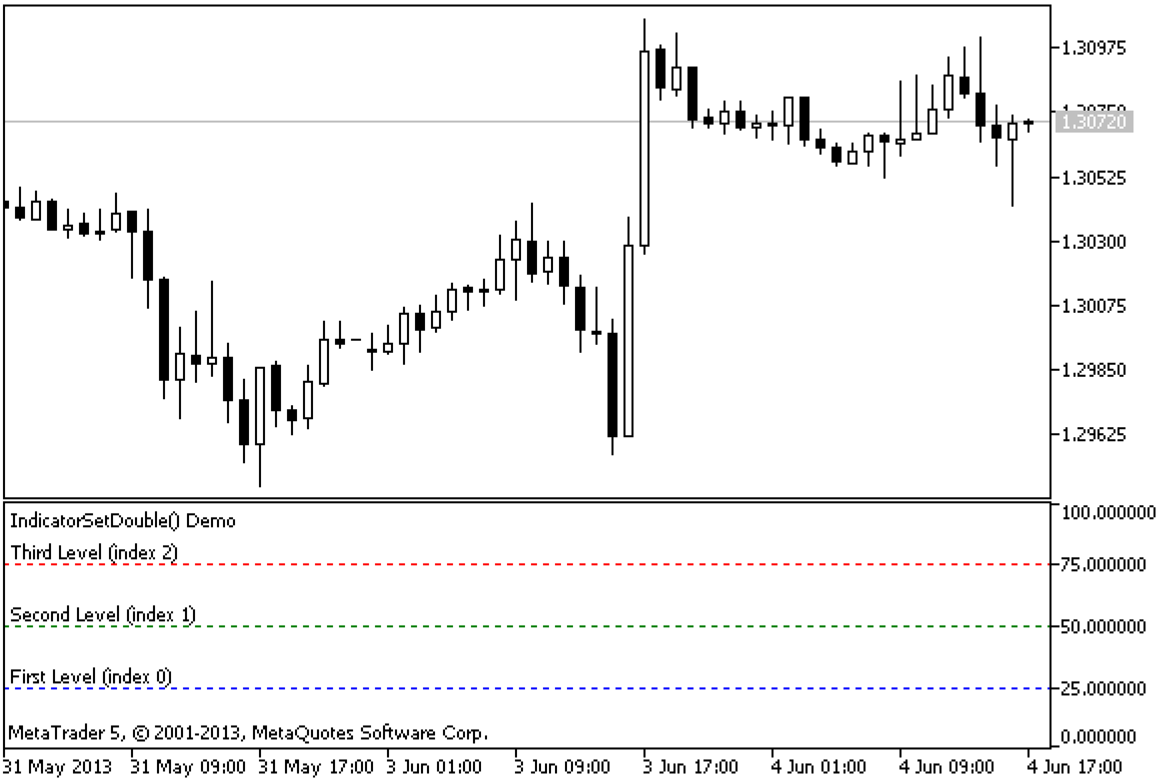
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, the first level value for the indicator in a separate subwindow can be set in two ways:

- property indicator_level**1** 50 - the value of 1 is used for specifying the level number,
- IndicatorSetDouble(INDICATOR_LEVELVALUE, **0**, 50) - 0 is used for specifying the first level.

Example: indicator that turns upside down the maximum and minimum values of the indicator window and values of levels on which the horizontal lines are placed.


1.30975
1.30720
1.30525
1.30300
1.30075
1.29850
1.29625
IndicatorSetDouble() Demo
Third Level (index 2)
Second Level (index 1)
First Level (index 0)
MetaTrader 5, © 2001-2013, MetaQuotes Software Corp.
100.000000
75.000000
50.000000
25.000000
0.000000
31 May 2013
31 May 09:00
31 May 17:00
3 Jun 01:00
3 Jun 09:00
3 Jun 17:00
4 Jun 01:00
4 Jun 09:00
4 Jun 17:00

```
#property indicator_separate_window
//--- set the maximum and minimum values for the indicator window
#property indicator_minimum  0
#property indicator_maximum  100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
//--- set thickness of horizontal levels
#property indicator_levelwidth 1
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- set color for each level
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,clrGreen);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,2,clrRed);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
   static double level1=25,level2=50,level3=75;
   static double max=100,min=0, shift=100;
//--- calculate ticks
   tick_counter++;
//--- turn levels upside down on every 10th tick
   if(tick_counter%10==0)
     {
      //--- invert sign for the level values
      level1=-level1;
      level2=-level2;
      level3=-level3;
      //--- invert sign for the maximum and minimum values
      max-=shift;
      min-=shift;
      //--- invert the shift value
      shift=-shift;
      //--- set new level values
      IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,2,level3);
      //--- set new values of maximum and minimum in the indicator window
      Print("Set up max = ",max,",   min = ",min);
      IndicatorSetDouble(INDICATOR_MAXIMUM,max);
      IndicatorSetDouble(INDICATOR_MINIMUM,min);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Indicator Styles in Examples, Connection between Indicator Properties and Functions, Drawing Styles

IndicatorSetInteger

The function sets the value of the corresponding indicator property. Indicator property must be of the int or color type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetInteger(
   int  prop_id,           // identifier
   int  prop_value         // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetInteger(
   int  prop_id,           // identifier
   int  prop_modifier,     // modifier
   int  prop_value         // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

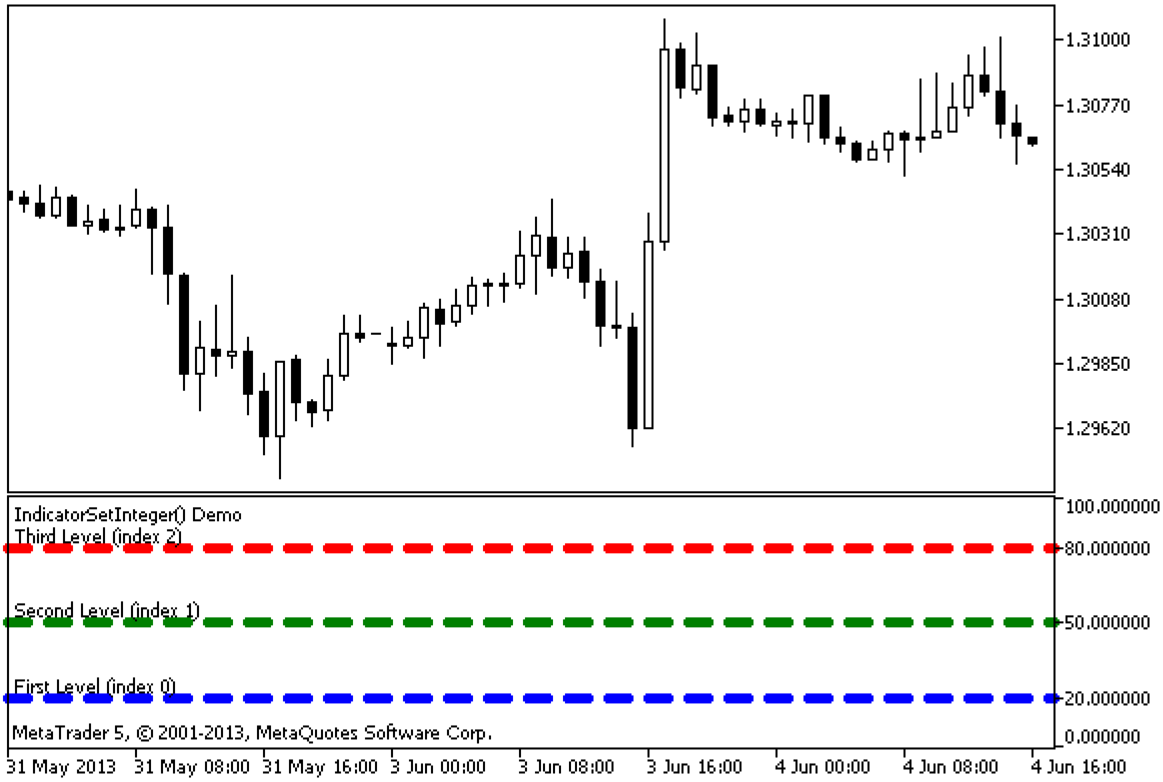
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, in order to set thickness of the first horizontal line use zeroth index:

- IndicatorSetInteger(INDICATOR_LEVELWIDTH, **0**, 5) - index 0 is used to set thickness of the first level.

Example: indicator that sets color, style and thickness of the indicator horizontal lines.


1.31000
1.30770
1.30540
1.30310
1.30080
1.29850
1.29620
IndicatorSetInteger() Demo
Third Level (index 2)
Second Level (index 1)
First Level (index 0)
MetaTrader 5, © 2001-2013, MetaQuotes Software Corp.
100.000000
80.000000
50.000000
20.000000
0.000000
31 May 2013
31 May 08:00
31 May 16:00
3 Jun 00:00
3 Jun 08:00
3 Jun 16:00
4 Jun 00:00
4 Jun 08:00
4 Jun 16:00

```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- set thickness of horizontal levels
#property indicator_levelwidth 5
//--- set color of horizontal levels
#property indicator_levelcolor clrAliceBlue
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetInteger() Demo");
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
//--- calculate ticks
   tick_counter++;
//--- and calculate colors of horizontal levels depending on the tick counter
   ChangeLevelColor(0,tick_counter,3,6,10); // three last parameters are switching the
   ChangeLevelColor(1,tick_counter,3,6,8);
   ChangeLevelColor(2,tick_counter,4,7,9);
//--- modify style of horizontal levels
   ChangeLevelStyle(0,tick_counter);
   ChangeLevelStyle(1,tick_counter+5);
   ChangeLevelStyle(2,tick_counter+15);
//--- get width as the remainder of integer division of the ticks number by 5
   int width=tick_counter%5;
//--- iterate over all horizontal levels and set thickness
   for(int l=0;l<3;l++)
      IndicatorSetInteger(INDICATOR_LEVELWIDTH,l,width+1);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Set color of horizontal line in the separate indicator window    |
//+------------------------------------------------------------------+
void ChangeLevelColor(int level,      // number of horizontal line
                      int tick_number,// dividend, number to get the remainder of div
                      int f_trigger,  // first divisor of color switching
                      int s_trigger,  // second divisor of color switching
                      int t_trigger)  // third divisor of color switching
  {
   static color colors[3]={clrRed,clrBlue,clrGreen};
//--- index of color from the colors[] array
   int index=-1;
//--- calculate the number of color from the colors[] array to paint horizontal line
   if(tick_number%f_trigger==0)
      index=0;   // if tick_number is divided by f_trigger without the remainder
   if(tick_number%s_trigger==0)
      index=1;   // if tick_number is divided by s_trigger without the remainder
   if(tick_number%t_trigger==0)
      index=2;   // if tick_number is divided by t_trigger without the remainder
//--- if color is defined, set it
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELCOLOR,level,colors[index]);
//---
  }
//+------------------------------------------------------------------+
//| Set style of horizontal line in the separate indicator window    |
//+------------------------------------------------------------------+
void ChangeLevelStyle(int level,     // number of horizontal line
                      int tick_number// number to get the remainder of division
                      )
  {
//--- array to store styles
   static ENUM_LINE_STYLE styles[5]=
     {STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- index of style from the styles[] array
   int index=-1;
//--- calculate the number from the styles[] array to set style of horizontal line
   if(tick_number%50==0)
      index=5;   // if tick_number is divided by 50 without the remainder, then style
   if(tick_number%40==0)
      index=4;   // ... style is STYLE_DASHDOT
   if(tick_number%30==0)
      index=3;   // ... STYLE_DOT
   if(tick_number%20==0)
      index=2;   // ... STYLE_DASH
   if(tick_number%10==0)
      index=1;   // ... STYLE_SOLID
//--- if style is defined, set it
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELSTYLE,level,styles[index]);
  }
```

See also

Custom Indicator Properties, Program Properties (#property), Drawing Styles

IndicatorSetString

The function sets the value of the corresponding indicator property. Indicator property must be of the string type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetString(
   int      prop_id,           // identifier
   string   prop_value         // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetString(
   int      prop_id,           // identifier
   int      prop_modifier,     // modifier
   string   prop_value         // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

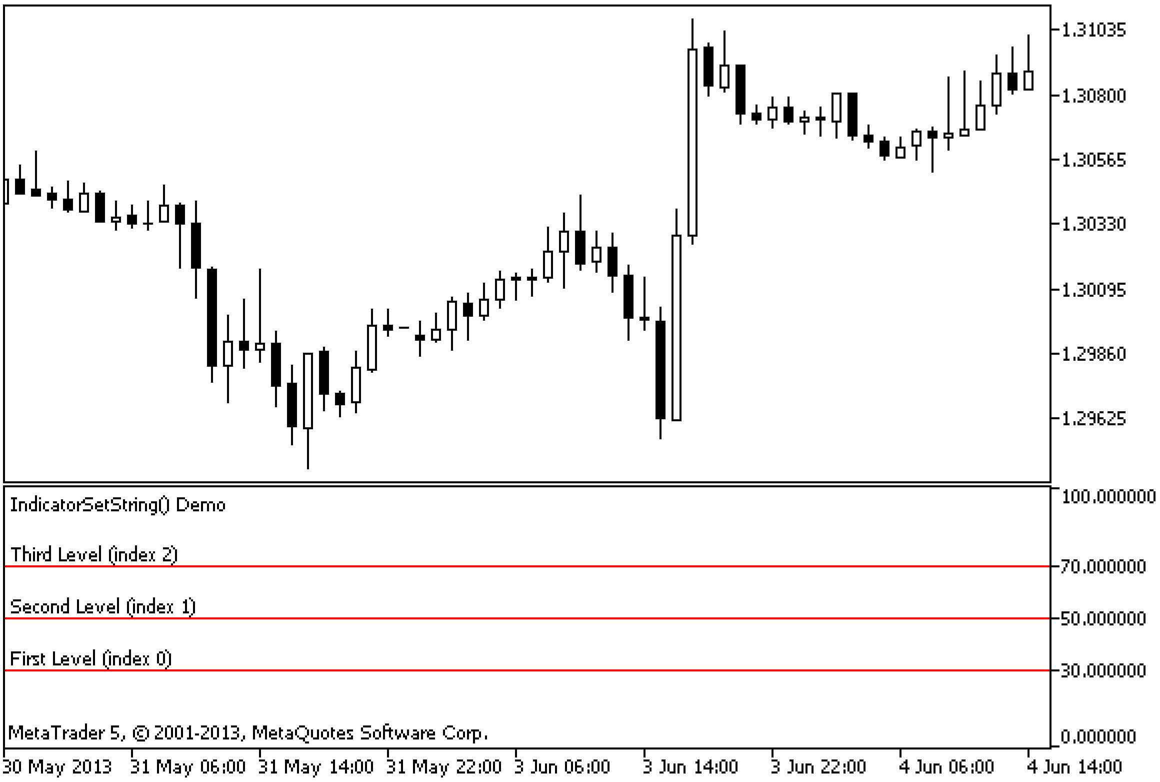
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, in order to set description of the first horizontal line use zeroth index:

- IndicatorSetString(INDICATOR_LEVELTEXT, **0**, "First Level") - index 0 is used to set text description of the first level.

Example: indicator that sets text labels to the indicator horizontal lines.


1.31035
1.30800
1.30565
1.30330
1.30095
1.29860
1.29625
IndicatorSetString() Demo
Third Level (index 2)
Second Level (index 1)
First Level (index 0)
MetaTrader 5, © 2001-2013, MetaQuotes Software Corp.
100.000000
70.000000
50.000000
30.000000
0.000000
30 May 2013
31 May 06:00
31 May 14:00
31 May 22:00
3 Jun 06:00
3 Jun 14:00
3 Jun 22:00
4 Jun 06:00
4 Jun 14:00

```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- set color of horizontal levels
#property indicator_levelcolor clrRed
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_SOLID
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetString() Demo");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---

//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Custom Indicator Properties, Program Properties (#property)

PlotIndexSetDouble

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the double type.

```
bool  PlotIndexSetDouble(
   int      plot_index,      // plotting style index
   int      prop_id,         // property identifier
   double   prop_value       // value to be set
   );
```

Parameters

plot_index

[in] Index of the graphical plotting

prop_id

[in] The value can be one of the values of the ENUM_PLOT_PROPERTY_DOUBLE enumeration.

prop_value

[in] The value of the property.

Return Value

If successful, returns true, otherwise false.

PlotIndexSetInteger

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the int, char, bool or color type. There are 2 variants of the function.

Call indicating identifier of the property.

```
bool  PlotIndexSetInteger(
   int  plot_index,         // plotting style index
   int  prop_id,            // property identifier
   int  prop_value          // value to be set
   );
```

Call indicating the identifier and modifier of the property.

```
bool  PlotIndexSetInteger(
   int  plot_index,         // plotting style index
   int  prop_id,            // property identifier
   int  prop_modifier,      // property modifier
   int  prop_value          // value to be set
   )
```

Parameters

plot_index

[in] Index of the graphical plotting

prop_id

[in] The value can be one of the values of the ENUM_PLOT_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. Only color index properties require a modifier.

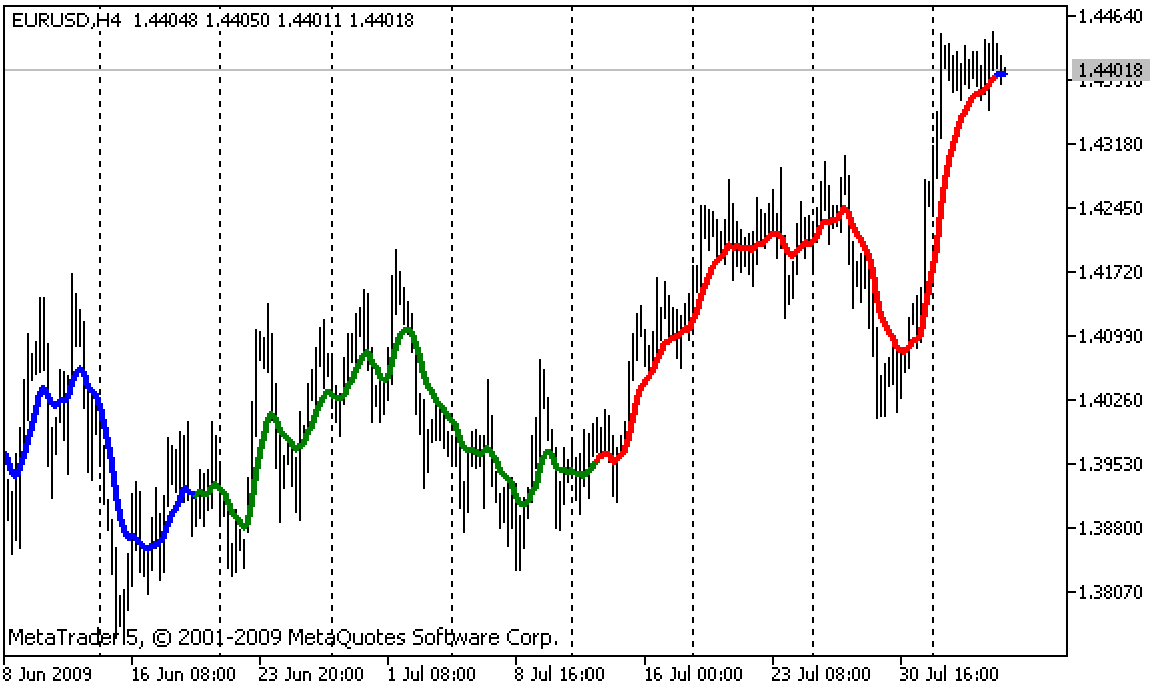
prop_value

[in] The value of the property.

Return Value

If successful, returns true, otherwise false.

Example: an indicator that draws a three-color line. The color scheme changes every 5 ticks.



```
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrGreen,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double         ColorLineBuffer[];
double         ColorBuffer[];
int            MA_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
   MA_handle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| get color index                                                  |
//+------------------------------------------------------------------+
int getIndexOfColor(int i)
  {
   int j=i%300;
   if(j<100) return(0);// first index
   if(j<200) return(1);// second index
   return(2); // third index
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   static int ticks=0,modified=0;
   int limit;
//--- first calculation or number of bars was changed
   if(prev_calculated==0)
     {
      //--- copy values of MA into indicator buffer ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);// copying failed - throw away
      //--- now set line color for every bar
      for(int i=0;i<rates_total;i++)
         ColorBuffer[i]=getIndexOfColor(i);
     }
   else
     {
      //--- copy values of MA into indicator buffer ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);

      ticks++;// ticks counting
      if(ticks>=5)//it's time to change color scheme
        {
         ticks=0; // reset counter
         modified++; // counter of color changes
         if(modified>=3)modified=0;// reset counter
         ResetLastError();
         switch(modified)
           {
            case 0:// first color scheme
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrRed);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrBlue);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrGreen);
               Print("Color scheme "+modified);
               break;
            case 1:// second color scheme
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrYellow);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrPink);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLightSlateGray);
               Print("Color scheme "+modified);
               break;
            default:// third color scheme
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrLightGoldenrod);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrOrchid);
               PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLimeGreen);
               Print("Color scheme "+modified);
           }
        }
      else
        {
         //--- set start position
         limit=prev_calculated-1;
         //--- now we set line color for every bar
         for(int i=limit;i<rates_total;i++)
            ColorBuffer[i]=getIndexOfColor(i);
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

PlotIndexSetString

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the string type.

```
bool  PlotIndexSetString(
   int      plot_index,     // plotting style index
   int      prop_id,        // property identifier
   string   prop_value      // value to be set
   );
```

Parameters

plot_index

[in] Index of graphical plot

prop_id

[in] The value can be one of the values of the ENUM_PLOT_PROPERTY_STRING enumeration.

prop_value

[in] The value of the property.

Return Value

If successful, returns true, otherwise false.

PlotIndexGetInteger

The function sets the value of the corresponding property of the corresponding indicator line. The indicator property must be of the int, color, bool or char type. There are 2 variants of the function.

Call indicating identifier of the property.

```
int  PlotIndexGetInteger(
   int  plot_index,         // plotting style index
   int  prop_id,            // property identifier
   );
```

Call indicating the identifier and modifier of the property.

```
int  PlotIndexGetInteger(
   int  plot_index,         // plotting index
   int  prop_id,            // property identifier
   int  prop_modifier       // property modifier
   )
```

Parameters

plot_index

[in] Index of the graphical plotting

prop_id

[in] The value can be one of the values of the ENUM_PLOT_PROPERTY_INTEGER enumeration.

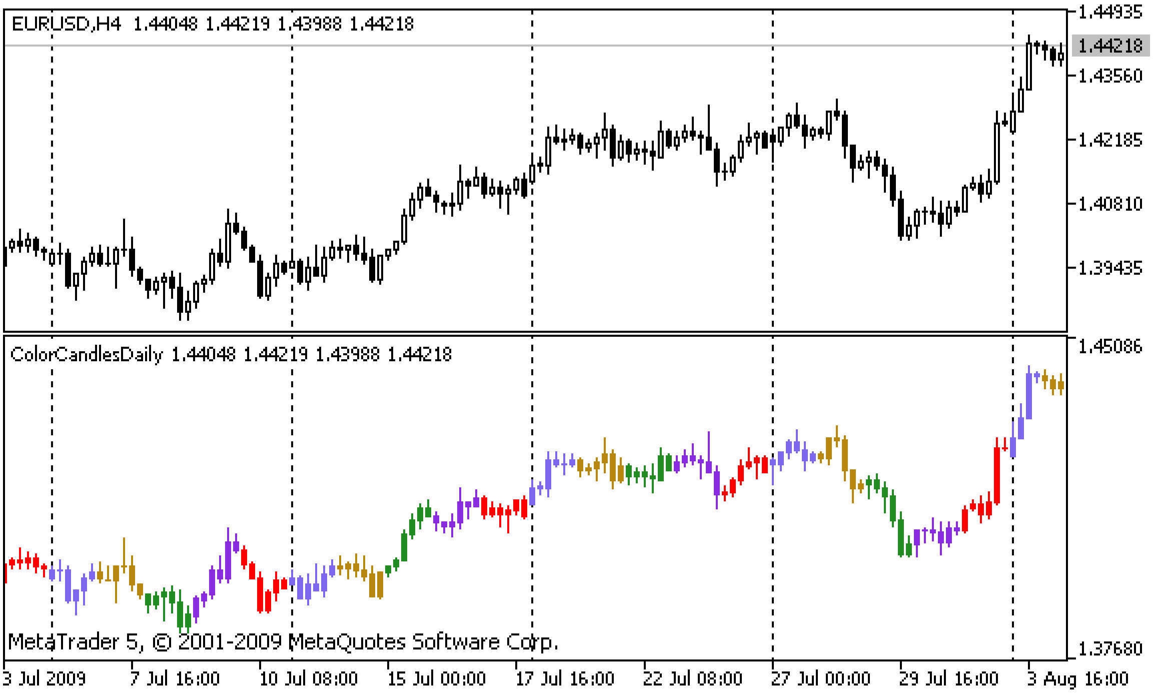
prop_modifier

[in] Modifier of the specified property. Only color index properties require a modifier.

Note

Function is designed to extract the settings of drawing of the appropriate indicator line. The function works in tandem with the function PlotIndexSetInteger to copy the drawing properties of one line to another.

Example: an indicator that colors candles depending on the day of the week. Colors for each day are set in a programmatically.



```
#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         OpenBuffer[];
double         HighBuffer[];
double         LowBuffer[];
double         CloseBuffer[];
double         ColorCandlesColors[];
color          ColorOfDay[6]={CLR_NONE,clrMediumSlateBlue,
                              clrDarkGoldenrod,clrForestGreen,clrBlueViolet,clrRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,OpenBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,HighBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LowBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,CloseBuffer,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- set number of colors in color buffer
```

```
   PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,6);
//--- set colors for color buffer
   for(int i=1;i<6;i++)
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,i,ColorOfDay[i]);
//--- set accuracy
   IndicatorSetInteger(INDICATOR_DIGITS,_Digits);
   printf("We have %u colors of days",PlotIndexGetInteger(0,PLOT_COLOR_INDEXES));
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   int i;
   MqlDateTime t;
//----
   if(prev_calculated==0) i=0;
   else i=prev_calculated-1;
//----
   while(i<rates_total)
     {
      OpenBuffer[i]=open[i];
      HighBuffer[i]=high[i];
      LowBuffer[i]=low[i];
      CloseBuffer[i]=close[i];
      //--- set color for every candle
      TimeToStruct(time[i],t);
      ColorCandlesColors[i]=t.day_of_week;
      //---
      i++;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

Object Functions

This is the group of functions intended for working with graphic objects relating to any specified chart.

The functions defining the properties of graphical objects, as well as ObjectCreate() and ObjectMove() operations for creating and moving objects along the chart are actually used for sending commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. Visual changes in the properties of graphical objects are implemented when handling the queue of the chart events.

Thus, do not expect an immediate visual update of graphical objects after calling these functions. Generally, the graphical objects on the chart are updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use ChartRedraw() function to forcefully update the graphical objects.

Function	Action
ObjectCreate	Creates an object of the specified type in a specified chart
ObjectName	Returns the name of an object of the corresponding type in the specified chart (specified chart subwindow)
ObjectDelete	Removes the object with the specified name from the specified chart (from the specified chart subwindow)
ObjectsDeleteAll	Removes all objects of the specified type from the specified chart (from the specified chart subwindow)
ObjectFind	Searches for an object with the specified ID by the name
ObjectGetTimeByValue	Returns the time value for the specified object price value
ObjectGetValueByTime	Returns the price value of an object for the specified time
ObjectMove	Changes the coordinates of the specified object anchor point
ObjectsTotal	Returns the number of objects of the specified type in the specified chart (specified chart subwindow)
ObjectGetDouble	Returns the double value of the corresponding object property
ObjectGetInteger	Returns the integer value of the corresponding object property
ObjectGetString	Returns the string value of the corresponding object property
ObjectSetDouble	Sets the value of the corresponding object property
ObjectSetInteger	Sets the value of the corresponding object property
ObjectSetString	Sets the value of the corresponding object property
TextSetFont	Sets the font for displaying the text using drawing methods (Arial 20 used by default)
TextOut	Transfers the text to the custom array (buffer) designed for creation of a graphical resource

Function	Action
TextGetSize	Returns the string's width and height at the current font settings

Every graphical object should have a name unique within one chart, including its subwindows. Changing of a name of a graphic object generates two events: event of deletion of an object with the old name, and event of creation of an object with a new name.

After an object is created or an object property is modified it is recommended to call the ChartRedraw() function, which commands the client terminal to forcibly draw a chart (and all visible objects in it).

ObjectCreate

The function creates an object with the specified name, type, and the initial coordinates in the specified chart subwindow. During creation up to 30 coordinates can be specified.

```
bool  ObjectCreate(
   long         chart_id,      // chart identifier
   string       name,          // object name
   ENUM_OBJECT  type,          // object type
   sub_window   nwin,          // window index
   datetime     time1,         // time of the first anchor point
   double       price1,        // price of the first anchor point
   ...
   datetime     timeN=0,       // time of the N-th anchor point
   double       priceN=0,      // price of the N-th anchor point
   ...
   datetime     time30=0,      // time of the 30th anchor point
   double       price30=0      // price of the 30th anchor point
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object. The name must be unique within a chart, including its subwindows.

type

[in] Object type. The value can be one of the values of the ENUM_OBJECT enumeration.

sub_window

[in] Number of the chart subwindow. 0 means the main chart window. The specified subwindow must exist, otherwise the function returns false.

time1

[in] The time coordinate of the first anchor.

price1

[in] The price coordinate of the first anchor point.

timeN=0

[in] The time coordinate of the N-th anchor point.

priceN=0

[in] The price coordinate of the N-th anchor point.

time30=0

[in] The time coordinate of the thirtieth anchor point.

price30=0

[in] The price coordinate of the thirtieth anchor point.

Return Value

The function returns true if the command has been successfully added to the queue of the specified chart, or false otherwise. If an object has already been created, an attempt is made to change its coordinates.

Note

An asynchronous call is always used for ObjectCreate(), that is why the function only returns the results of adding the command to a chart queue. In this case, true only means that the command has been successfully enqueued, but the result of its execution is unknown.

To check the command execution result, you can use the ObjectFind() function or any other function that requests object properties, such as ObjectGetXXX. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result (due to the synchronous call), and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

An object name should not exceed 63 characters.

The numbering of the chart subwindows (if there are subwindows with indicators in the chart) starts with 1. The main chart window of the chart is and always has index 0.

The large number of anchor points (up to 30) is implemented for future use. At the same time, the limit of 30 possible anchor points for graphical objects is determined by the limit on the number of parameters (not more than 64) that can be used when calling a function.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

There is a certain number of anchor points that must be specified when creating each object type:

ID	Description	Anchor Points
OBJ_VLINE	Vertical Line	One anchor point. Actually only the time coordinate is used.
OBJ_HLINE	Horizontal Line	One anchor point. Actually only the price coordinate is used.
OBJ_TREND	Trend Line	Two anchor points.
OBJ_TRENDBYANGLE	Trend Line By Angle	Two anchor points.
OBJ_CYCLES	Cycle Lines	Two anchor points.
OBJ_ARROWED_LINE	Arrowed Line	Two anchor points.
OBJ_CHANNEL	Equidistant Channel	Three anchor points.
OBJ_STDDEVCHANNEL	Standard Deviation Channel	Two anchor points.
OBJ_REGRESSION	Linear Regression Channel	Two anchor points.

ID	Description	Anchor Points
OBJ_PITCHFORK	Andrews' Pitchfork	Three anchor points.
OBJ_GANNLINE	Gann Line	Two anchor points.
OBJ_GANNFAN	Gann Fan	Two anchor points.
OBJ_GANNGRID	Gann Grid	Two anchor points.
OBJ_FIBO	Fibonacci Retracement	Two anchor points.
OBJ_FIBOTIMES	Fibonacci Time Zones	Two anchor points.
OBJ_FIBOFAN	Fibonacci Fan	Two anchor points.
OBJ_FIBOARC	Fibonacci Arcs	Two anchor points.
OBJ_FIBOCHANNEL	Fibonacci Channel	Three anchor points.
OBJ_EXPANSION	Fibonacci Expansion	Three anchor points.
OBJ_ELLIOTWAVE5	Elliott Motive Wave	Five anchor points.
OBJ_ELLIOTWAVE3	Elliott Correction Wave	Three anchor points.
OBJ_RECTANGLE	Rectangle	Two anchor points.
OBJ_TRIANGLE	Triangle	Three anchor points.
OBJ_ELLIPSE	Ellipse	Three anchor points.
OBJ_ARROW_THUMB_UP	Thumbs Up	One anchor point.
OBJ_ARROW_THUMB_DOWN	Thumbs Down	One anchor point.
OBJ_ARROW_UP	Arrow Up	One anchor point.
OBJ_ARROW_DOWN	Arrow Down	One anchor point.
OBJ_ARROW_STOP	Stop Sign	One anchor point.
OBJ_ARROW_CHECK	Check Sign	One anchor point.
OBJ_ARROW_LEFT_PRICE	Left Price Label	One anchor point.
OBJ_ARROW_RIGHT_PRICE	Right Price Label	One anchor point.
OBJ_ARROW_BUY	Buy Sign	One anchor point.
OBJ_ARROW_SELL	Sell Sign	One anchor point.
OBJ_ARROW	Arrow	One anchor point.
OBJ_TEXT	Text	One anchor point.
OBJ_LABEL	Label	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.

ID	Description	Anchor Points
OBJ_BUTTON	Button	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.
OBJ_CHART	Chart	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.
OBJ_BITMAP	Bitmap	One anchor point.
OBJ_BITMAP_LABEL	Bitmap Label	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.
OBJ_EDIT	Edit	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.
OBJ_EVENT	The "Event" object corresponding to an event in the economic calendar	One anchor point. Actually only the time coordinate is used.
OBJ_RECTANGLE_LABEL	The "Rectangle label" object for creating and designing the custom graphical interface.	Position is set using the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.

ObjectName

The function returns the name of the corresponding object in the specified chart, in the specified subwindow, of the specified type.

```
string  ObjectName(
   long   chart_id,            // chart identifier
   int    pos,                 // number in the list of objects
   int    sub_window=-1,       // window index
   int    type=-1              // object type
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

pos

[in] Ordinal number of the object according to the specified filter by the number and type of the subwindow.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the ENUM_OBJECT enumeration. -1 means all types.

Return Value

Name of the object is returned in case of success.

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectDelete

The function removes the object with the specified name from the specified chart.

```
bool  ObjectDelete(
   long     chart_id,      // chart identifier
   string   name           // object name
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of object to be deleted.

Return Value

The function returns true if the command has been successfully added to the queue of the specified chart, or false otherwise.

Note

An asynchronous call is always used for ObjectDelete(), that is why the function only returns the results of adding the command to a chart queue. In this case, true only means that the command has been successfully enqueued, but the result of its execution is unknown.

To check the command execution result, you can use the ObjectFind() function or any other function that requests object properties, such as ObjectGetXXX. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result (due to the synchronous call), and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectsDeleteAll

Removes all objects from the specified chart, specified chart subwindow, of the specified type.

```
int  ObjectsDeleteAll(
   long   chart_id,            // chart identifier
   int    sub_window=-1,       // window index
   int    type=-1              // object type
   );
```

Removes all objects of the specified type using prefix in object names.

```
int  ObjectsDeleteAll(
   long            chart_id,    // chart ID
   const string      prefix,    // prefix in object name
   int          sub_window=-1,  // window index
   int        object_type=-1    // object type
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

prefix

[in] Prefix in object names. All objects whose names start with this set of characters will be removed from chart. You can specify prefix as 'name' or 'name*' – both variants will work the same. If an empty string is specified as the prefix, objects with all possible names will be removed.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the ENUM_OBJECT enumeration. -1 means all types.

Return Value

Returns the number of deleted objects. To read more about the error call GetLastError().

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

ObjectFind

The function searches for an object with the specified name in the chart with the specified ID.

```
int  ObjectFind(
   long     chart_id,      // chart identifier
   string   name           // object name
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] The name of the searched object.

Return Value

If successful the function returns the number of the subwindow (0 means the main window of the chart), in which the object is found. If the object is not found, the function returns a negative number. To read more about the error call GetLastError().

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectGetTimeByValue

The function returns the time value for the specified price value of the specified object.

```
datetime  ObjectGetTimeByValue(
   long     chart_id,      // chart identifier
   string   name,          // object name
   double   value,         // Price
   int      line_id        // Line number
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

value

[in] Price value.

line_id

[in] Line identifier.

Return Value

The time value for the specified price value of the specified object.

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLINE)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines
- Arrowed line (OBJ_ARROWED_LINE)

See also

Object Types

ObjectGetValueByTime

The function returns the price value for the specified time value of the specified object.

```
double  ObjectGetValueByTime(
   long       chart_id,      // chart identifier
   string     name,          // object name
   datetime   time,          // Time
   int        line_id        // Line number
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

time

[in] Time value.

line_id

[in] Line ID.

Return Value

The price value for the specified time value of the specified object.

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLINE)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines
- Arrowed line (OBJ_ARROWED_LINE)

See also

Object Types

ObjectMove

The function changes coordinates of the specified anchor point of the object.

```
bool  ObjectMove(
   long      chart_id,        // chart identifier
   string    name,            // object name
   int       point_index,     // anchor point number
   datetime  time,            // Time
   double    price            // Price
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

point_index

[in] Index of the anchor point. The number of anchor points depends on the type of object.

time

[in] Time coordinate of the selected anchor point.

price

[in] Price coordinate of the selected anchor point.

Return Value

The function returns true if the command has been successfully added to the queue of the specified chart, or false otherwise.

Note

An asynchronous call is always used for ObjectMove(), that is why the function only returns the results of adding the command to a chart queue. In this case, true only means that the command has been successfully enqueued, but the result of its execution is unknown.

To check the command execution result, you can use a function that requests object properties, such as ObjectGetXXX. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result (due to the synchronous call), and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

ObjectsTotal

The function returns the number of objects in the specified chart, specified subwindow, of the specified type.

```
int  ObjectsTotal(
   long   chart_id,            // chart identifier
   int    sub_window=-1,       // window index
   int    type=-1              // object type
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the ENUM_OBJECT enumeration. -1 means all types.

Return Value

The number of objects.

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

ObjectSetDouble

The function sets the value of the corresponding object property. The object property must be of the double type. There are 2 variants of the function.

Setting property value, without modifier

```
bool  ObjectSetDouble(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_DOUBLE     prop_id,           // property
   double                          prop_value         // value
   );
```

Setting a property value indicating the modifier

```
bool  ObjectSetDouble(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_DOUBLE     prop_id,           // property
   int                             prop_modifier,     // modifier
   double                          prop_value         // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_DOUBLE enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

Note

The function uses an asynchronous call, which means that the function does not wait for the execution of the command that has been added to the queue of the specified chart. Instead, it immediately returns control.

To check the command execution result, you can use a function that requests the specified object property. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result, and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

Example of creating a Fibonacci object and adding a new level in it

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- auxiliary arrays
   double high[],low[],price1,price2;
   datetime time[],time1,time2;
//--- Copy the open prices - 100 latest bars are enough
   int copied=CopyHigh(Symbol(),0,0,100,high);
   if(copied<=0)
     {
      Print("Failed to copy the values of the High price series");
      return;
     }
//--- Copy the close price - 100 latest bars are enough
   copied=CopyLow(Symbol(),0,0,100,low);
   if(copied<=0)
     {
      Print("Failed to copy the values of the Low price series");
      return;
     }
//--- Copy the open time for the last 100 bars
   copied=CopyTime(Symbol(),0,0,100,time);
   if(copied<=0)
     {
      Print("Failed to copy the values of the price series of Time");
      return;
     }
//--- Organize access to the copied data as to timeseries - backwards
   ArraySetAsSeries(high,true);
   ArraySetAsSeries(low,true);
   ArraySetAsSeries(time,true);

//--- Coordinates of the first anchor point of the Fibo object
   price1=high[70];
   time1=time[70];
//--- Coordinates of the second anchor point of the Fibo object
   price2=low[50];
   time2=time[50];

//--- Time to create the Fibo object
   bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2);
   if(created) // If the object is created successfully
     {
      //--- set the color of Fibo levels
      ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
      //--- by the way, how much Fibo levels do we have?
      int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
      Print("Fibo levels before = ",levels);
      //---output to the Journal => number of level:value level_desription
      for(int i=0;i<levels;i++)
        {
         Print(i,": ",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
               "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
        }
      //--- Try to increase the number of levels per unit
      bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
      if(!modified) // failed to change the number of levels
        {
         Print("Failed to change the number of levels of Fibo, error ",GetLastError())
        }
      //--- just inform
      Print("Fibo levels after = ",ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS));
      //--- set a value for a newly created level
      bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
      if(added) // managed to set a value for the level
        {
         Print("Successfully set one more Fibo level");
         //--- Also do not forget to set the level description
         ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
         ChartRedraw(0);
         //--- Get the actual value of the number of levels in the Fibo object
         levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
         Print("Fibo levels after adding = ",levels);
         //--- once again output all levels - just to make sure
         for(int i=0;i<levels;i++)
           {
            Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                  "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
           }
        }
      else // Fails if you try to increase the number of levels in the Fibo object
        {
         Print("Failed to set one more Fibo level. Error ",GetLastError());
        }
     }
  }
```

See also

Object Types, Object Properties

ObjectSetInteger

The function sets the value of the corresponding object property. The object property must be of the datetime, int, color, bool or char type. There are 2 variants of the function.

Setting property value, without modifier

```
bool  ObjectSetInteger(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_INTEGER    prop_id,           // property
   long                            prop_value         // value
   );
```

Setting a property value indicating the modifier

```
bool  ObjectSetInteger(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_INTEGER    prop_id,           // property
   int                             prop_modifier,     // modifier
   long                            prop_value         // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

Note

The function uses an asynchronous call, which means that the function does not wait for the execution of the command that has been added to the queue of the specified chart. Instead, it immediately returns control.

To check the command execution result, you can use a function that requests the specified object property. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result, and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

An example of how to create a table of Web colors

```
//+------------------------------------------------------------------+
//|                                              Table of Web Colors|
//|                         Copyright 2011, MetaQuotes Software Corp |
//|                                        https://www.metaquotes.net |
//+------------------------------------------------------------------+
#define X_SIZE 140      // width of an edit object
#define Y_SIZE 33       // height of an edit object
//+------------------------------------------------------------------+
//| Array of web colors                                              |
//+------------------------------------------------------------------+
color ExtClr[140]=
  {
   clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,clrBisque,clr
   clrBlue,clrBlueViolet,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,clrChocolat
   clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod,clrDarkGra
   clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed,clrDarkSal
   clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDeepPink,clrDe
   clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clrGainsboro,c
   clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clrIndianRed,c
   clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,clrLightCor
   clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmon,clrLightS
   clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen,clrLinen,c
   clrMediumAquamarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMediumSeaGree
   clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightBlue,clrMint
   clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOrangeRed,clr
   clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPuff,clrPeru,
   clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clrSandyBrown,
   clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpringGreen,cl
   clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteSmoke,clrYel
  };
//+------------------------------------------------------------------+
//| Creating and initializing an edit object                         |
//+------------------------------------------------------------------+
void CreateColorBox(int x,int y,color c)
  {
//--- generate a name for a new edit object
   string name="ColorBox_"+(string)x+"_"+(string)y;
//--- create a new edit object
   if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
```

```
      {
       Print("Cannot create: '",name,"'");
       return;
      }
//--- set coordinates, width and height
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x*X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y*Y_SIZE);
   ObjectSetInteger(0,name,OBJPROP_XSIZE,X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YSIZE,Y_SIZE);
//--- set text color
   if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
   else            ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
//--- set background color
   ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- set text
   ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create 7x20 table of colored edit objects
   for(uint i=0;i<140;i++)
      CreateColorBox(i%7,i/7,ExtClr[i]);
  }
```

See also

Object Types, Object Properties

ObjectSetString

The function sets the value of the corresponding object property. The object property must be of the string type. There are 2 variants of the function.

Setting property value, without modifier

```
bool  ObjectSetString(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_STRING     prop_id,           // property
   string                          prop_value         // value
   );
```

Setting a property value indicating the modifier

```
bool  ObjectSetString(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_STRING     prop_id,           // property
   int                             prop_modifier,     // modifier
   string                          prop_value         // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

Note

The function uses an asynchronous call, which means that the function does not wait for the execution of the command that has been added to the queue of the specified chart. Instead, it immediately returns control.

To check the command execution result, you can use a function that requests the specified object property. However, you should keep in mind that such functions are added to the end of the queue of that chart, and they wait for the execution result, and can therefore be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

ObjectGetDouble

The function returns the value of the corresponding object property. The object property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  ObjectGetDouble(
   long                          chart_id,          // chart identifier
   string                        name,              // object name
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // property identifier
   int                           prop_modifier=0    // property modifier, if require
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetDouble(
   long                          chart_id,          // chart identifier
   string                        name,              // object name
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,           // property identifier
   int                           prop_modifier,     // property modifier
   double&                       double_var         // here we accept the property
   );
```

Parameters

`chart_id`

[in] Chart identifier. 0 means the current chart.

`name`

[in] Name of the object.

`prop_id`

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_DOUBLE enumeration.

`prop_modifier`

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

`double_var`

[out] Variable of the double type that received the value of the requested property.

Return Value

Value of the double type for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the double_var variable, otherwise returns false. To read more about the error call GetLastError().

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

ObjectGetInteger

The function returns the value of the corresponding object property. The object property must be of the datetime, int, color, bool or char type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  ObjectGetInteger(
   long                             chart_id,          // chart identifier
   string                           name,              // object name
   ENUM_OBJECT_PROPERTY_INTEGER     prop_id,           // property identifier
   int                              prop_modifier=0    // property modifier, if requi
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetInteger(
   long                             chart_id,          // chart identifier
   string                           name,              // object name
   ENUM_OBJECT_PROPERTY_INTEGER     prop_id,           // property identifier
   int                              prop_modifier,     // property modifier
   long&                            long_var           // here we accept the property
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

long_var

[out] Variable of the long type that receives the value of the requested property.

Return Value

The long value for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the long_var variable, otherwise returns false. To read more about the error call GetLastError().

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

ObjectGetString

The function returns the value of the corresponding object property. The object property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  ObjectGetString(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_STRING     prop_id,           // property identifier
   int                             prop_modifier=0    // property modifier, if require
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetString(
   long                            chart_id,          // chart identifier
   string                          name,              // object name
   ENUM_OBJECT_PROPERTY_STRING     prop_id,           // property identifier
   int                             prop_modifier,     // property modifier
   string&                         string_var         // here we accept the property 
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

string_var

[out] Variable of the string type that receives the value of the requested properties.

Return Value

String value for the first version of the call.

For the second version of the call returns true, if this property is maintained and the value has been placed into the string_var variable, otherwise returns false. To read more about the error call GetLastError().

Note

The function uses a synchronous call, which means that the function waits for the execution of all commands that have been enqueued for this chart prior to its call, that is why this function can be time consuming. This feature should be taken into account when working with a large number of objects on a chart.

When an object is renamed, two events are formed simultaneously. These events can be handled in an Expert Advisor or indicator by the OnChartEvent() function:

- an event of deletion of an object with the old name;
- an event of creation of an object with a new name.

TextSetFont

The function sets the font for displaying the text using drawing methods and returns the result of that operation. Arial font with the size -120 (12 pt) is used by default.

```
bool  TextSetFont(
   const string  name,            // font name or path to font file on the disk
   int           size,            // font size
   uint          flags,           // combination of flags
   int           orientation=0    // text slope angle
   );
```

Parameters

name

[in] Font name in the system or the name of the resource containing the font or the path to font file on the disk.

size

[in] The font size that can be set using positive and negative values. In case of positive values, the size of a displayed text does not depend on the operating system's font size settings. In case of negative values, the value is set in tenths of a point and the text size depends on the operating system settings ("standard scale" or "large scale"). See the Note below for more information about the differences between the modes.

flags

[in] Combination of flags describing font style.

orientation

[in] Text's horizontal inclination to X axis, the unit of measurement is 0.1 degrees. It means that `orientation=450 stands for inclination equal to 45 degrees.`

Return Value

Returns true if the current font is successfully installed, otherwise false. Possible code errors:

- ERR_INVALID_PARAMETER(4003) - *name* presents NULL or "" (empty string),
- ERR_INTERNAL_ERROR(4001) - operating system error (for example, an attempt to create a non-existent font).

Note

If "::" is used in font name, the font is downloaded from EX5 resource. If *name* font name is specified with an extension, the font is downloaded from the file, if the path starts from "\" or "/", the file is searched relative to MQL5 directory. Otherwise, it is searched relative to the path of EX5 file which called TextSetFont() function.

The font size is set using positive or negative values. This fact defines the dependence of the text size from the operating system settings (size scale).

- If the size is specified using a positive number, this size is transformed into physical measurement units of a device (pixels) when changing a logical font into a physical one, and this size corresponds to the height of the symbol glyphs picked from the available fonts. This case is not recommended when the texts displayed by TextOut() function and the ones displayed by OBJ_LABEL ("Label") graphical object are to be used together on the chart.

- If the size is specified using a negative number, this number is supposed to be set in tenths of a logical point and is divided by 10 (for example, -350 is equal to 35 logical points). An obtained value is then transformed into physical measurement units of a device (pixels) and corresponds to the absolute value of the height of a symbol picked from the available fonts. Multiply the font size specified in the object properties by -10 to make the size of a text on the screen similar to the one in OBJ_LABEL object.

The flags can be used as the combination of style flags with one of the flags specifying the font width. Flag names are shown below.

Flags for specifying font style

Flag	Description
FONT_ITALIC	Italic
FONT_UNDERLINE	Underline
FONT_STRIKEOUT	Strikeout

Flags for specifying font width

Flag
FW_DONTCARE
FW_THIN
FW_EXTRALIGHT
FW_ULTRALIGHT
FW_LIGHT
FW_NORMAL
FW_REGULAR
FW_MEDIUM
FW_SEMIBOLD
FW_DEMIBOLD
FW_BOLD
FW_EXTRABOLD
FW_ULTRABOLD
FW_HEAVY
FW_BLACK

See also

Resources, ResourceCreate(), ResourceSave(), TextOut()

TextOut

The function displays a text in a custom array (buffer) and returns the result of that operation. The array is designed to create the graphical resource.

```
bool  TextOut(
   const string       text,          // displayed text
   int                x,             // X coordinate
   int                y,             // Y coordinate
   uint               anchor,        // anchor type
   uint               &data[],       // output buffer
   uint               width,         // buffer width in pixels
   uint               height,        // buffer height in pixels
   uint               color,         // text color
   ENUM_COLOR_FORMAT  color_format   // color format for output
   );
```

Parameters

text

[in] Displayed text that will be written to the buffer. Only one-lined text is displayed.

x

[in] X coordinate of the anchor point of the displayed text.

y

[in] Y coordinate of the anchor point of the displayed text.

anchor

[in] The value out of the 9 pre-defined methods of the displayed text's anchor point location. The value is set by a combination of two flags – flags of horizontal and vertical text align. Flag names are listed in the Note below.

data[]

[in] Buffer, in which text is displayed. The buffer is used to create the graphical resource.

width

[in] Buffer width in pixels.

height

[in] Buffer height in pixels.

color

[in] Text color.

color_format

[in] Color format is set by ENUM_COLOR_FORMAT enumeration value.

Return Value

Returns true if successful, otherwise false.

Note

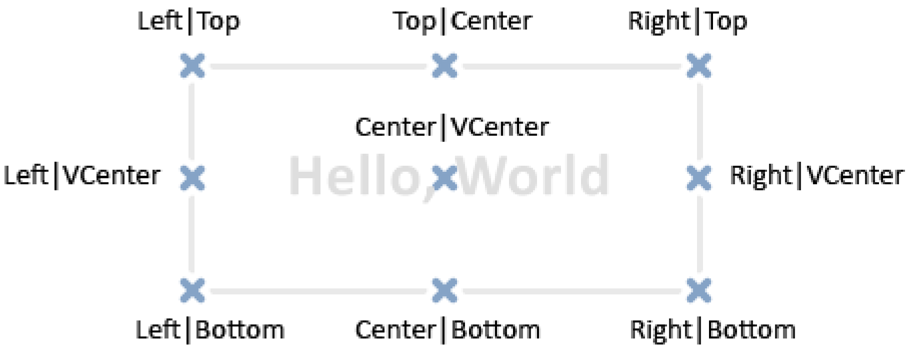
Anchor point specified by *anchor* is a combination of two flags of horizontal and vertical text align. Horizontal text align flags:

- TA_LEFT – anchor point on the left side of the bounding box
- TA_CENTER – horizontal anchor point is located at the center of the bounding box
- TA_RIGHT – anchor point on the right side of the bounding box

Vertical text align flags:

- TA_TOP – anchor point at the upper side of the bounding box
- TA_VCENTER – vertical anchor point is located at the center of the bounding box
- TA_BOTTOM – anchor point at the lower side of the bounding box

Possible combinations of flags and specified anchor points are shown in the image.



Example:

```
//--- width and height of the canvas (used for drawing)
#define IMG_WIDTH  200
#define IMG_HEIGHT 200
//--- display the parameter window before launching the script
#property script_show_inputs
//--- enable to set color format
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- drawing array (buffer)
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create OBJ_BITMAP_LABEL object for drawing
   ObjectCreate(0,"CLOCK",OBJ_BITMAP_LABEL,0,0,0);
//--- specify the name of the graphical resource for writing in CLOCK object
   ObjectSetString(0,"CLOCK",OBJPROP_BMPFILE,"::IMG");

//--- auxiliary variables
   double a;            // arrow corner
   uint   nm=2700;      // minute corner
   uint   nh=2700*12;   // hour counter
   uint   w,h;          // variables for receiving text string sizes
   int    x,y;           // variables for calculation of the current coordinates of te

//--- rotate clock hands in an infinite loop, till the script is stopped
   while(!IsStopped())
     {
      //--- clear the clock drawing buffer array
      ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
      //--- set the font for drawing digits for the clock-face
      TextSetFont("Arial",-200,FW_EXTRABOLD,0);
      //--- draw the clock-face
      for(int i=1;i<=12;i++)
        {
         //--- receive the size of the current hour on the clock-face
         TextGetSize(string(i),w,h);
         //--- calculate the coordinates of the current hour on the clock-face
         a=-((i*300)%3600*M_PI)/1800.0;
         x=IMG_WIDTH/2-int(sin(a)*80+0.5+w/2);
         y=IMG_HEIGHT/2-int(cos(a)*80+0.5+h/2);
         //--- output the hour on the clock-face to ExtImg[] buffer
         TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,
        }
      //--- now, specify the font for drawing the minute hand
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nm%3600));
      //--- receive the size of the minute hand
      TextGetSize("----->",w,h);
      //--- calculate the coordinates of the minute hand on the clock-face
      a=-(nm%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- output of the minute hand to the clock-face in ExtImg[]buffer
      TextOut("----->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_

      //--- now, set the font for drawing the minute hand
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nh/12%3600));
      TextGetSize("==>",w,h);
      //--- calculate the coordinates of the hour hand on the clock-face
      a=-(nh/12%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- output of the hour hand on the clock-face to ExtImg[] buffer
      TextOut("==>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_for

      //--- update the graphical resource
      ResourceCreate("::IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr_format);
      //--- forced chart update
      ChartRedraw();

      //--- increase hour and minute counters
      nm+=60;
      nh+=60;
      //--- keeping a short pause between the frames
      Sleep(10);
     }
//--- delete CLOCK object when completing the script's operation
   ObjectDelete(0,"CLOCK");
//---
  }
```

See also

Resources, ResourceCreate(), ResourceSave(), TextGetSize(), TextSetFont()

TextGetSize

The function returns the line width and height at the current font settings.

```
bool  TextGetSize(
   const string       text,          // text string
   uint&               width,         // buffer width in pixels
   uint&               height         // buffer height in pixels
   );
```

Parameters

text

[in] String, for which length and width should be obtained.

width

[out] Input parameter for receiving width.

height

[out] Input parameter for receiving height.

Return Value

Returns true if successful, otherwise false. Possible code errors:

- ERR_INTERNAL_ERROR(4001) - operating system error.

See also

Resources, ResourceCreate(), ResourceSave(), TextSetFont(), TextOut()

Technical Indicator Functions

All functions like iMA, iAC, iMACD, iIchimoku etc. create a copy of the corresponding technical indicator in the global cache of the client terminal. If a copy of the indicator with such parameters already exists, the new copy is not created, and the counter of references to the existing copy increases.

These functions return the handle of the appropriate copy of the indicator. Further, using this handle, you can receive data calculated by the corresponding indicator. The corresponding buffer data (technical indicators contain calculated data in their internal buffers, which can vary from 1 to 5, depending on the indicator) can be copied to a mql5-program using the CopyBuffer() function.

You can't refer to the indicator data right after it has been created, because calculation of indicator values requires some time, so it's better to create indicator handles in OnInit(). Function iCustom() creates the corresponding custom indicator, and returns its handle in case it is successfully create. Custom indicators can contain up to 512 indicator buffers, the contents of which can also be obtained by the CopyBuffer() function, using the obtained handle.

There is a universal method for creating any technical indicator using the IndicatorCreate() function. This function accepts the following data as input parameters:

- symbol name;
- timeframe;
- type of the indicator to create;
- number of input parameters of the indicator;
- an array of MqlParam type containing all the necessary input parameters.

The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note. Repeated call of the indicator function with the same parameters within one mql5-program does not lead to a multiple increase of the reference counter; the counter will be increased only once by 1. However, it's recommended to get the indicators handles in function OnInit() or in the class constructor, and further use these handles in other functions. The reference counter decreases when a mql5-program is deinitialized.

All indicator functions have at least 2 parameters - symbol and period. The NULL value of the symbol means the current symbol, the 0 value of the period means the current timeframe.

Function	Returns the handle of the indicator:
iAC	Accelerator Oscillator
iAD	Accumulation/Distribution
iADX	Average Directional Index
iADXWilder	Average Directional Index by Welles Wilder
iAlligator	Alligator
iAMA	Adaptive Moving Average

Function	Returns the handle of the indicator:
iAO	Awesome Oscillator
iATR	Average True Range
iBearsPower	Bears Power
iBands	Bollinger Bands®
iBullsPower	Bulls Power
iCCI	Commodity Channel Index
iChaikin	Chaikin Oscillator
iCustom	Custom indicator
iDEMA	Double Exponential Moving Average
iDeMarker	DeMarker
iEnvelopes	Envelopes
iForce	Force Index
iFractals	Fractals
iFrAMA	Fractal Adaptive Moving Average
iGator	Gator Oscillator
iIchimoku	Ichimoku Kinko Hyo
iBWMFI	Market Facilitation Index by Bill Williams
iMomentum	Momentum
iMFI	Money Flow Index
iMA	Moving Average
iOsMA	Moving Average of Oscillator (MACD histogram)
iMACD	Moving Averages Convergence-Divergence
iOBV	On Balance Volume
iSAR	Parabolic Stop And Reverse System
iRSI	Relative Strength Index
iRVI	Relative Vigor Index
iStdDev	Standard Deviation
iStochastic	Stochastic Oscillator
iTEMA	Triple Exponential Moving Average
iTriX	Triple Exponential Moving Averages Oscillator

Function	Returns the handle of the indicator:
iWPR	Williams' Percent Range
iVIDyA	Variable Index Dynamic Average
iVolumes	Volumes

iAC

The function creates Accelerator Oscillator in a global cache of the client terminal and returns its handle. It has only one buffer.

```
int  iAC(
   string           symbol,      // symbol name
   ENUM_TIMEFRAMES  period       // period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES enumeration values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAC.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAC technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plotting of iAC
#property indicator_label1  "iAC"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAC,                // use iAC
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iAC;          // type of the function
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffers
double iACBuffer[];
double iACColors[];
//--- variable for storing the handle of the iAC indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Accelerator Oscillator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,iACBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iACColors,INDICATOR_COLOR_INDEX);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iAC)
      handle=iAC(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AC);
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
```

```
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iAC indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Accelerator Oscillator indicator is calculated fo
   short_name=StringFormat("iAC(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iAC indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iACBuffer array is greater than the number of values in the iAC in
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iACBuffer and iACColors arrays with values from the Accelerator Oscilla
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffer(iACBuffer,iACColors,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Accelerator Oscillator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iAC indicator                 |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],        // indicator buffer of Accelerator
                          double &color_indexes[], // color buffer (for storing of co
                          int ind_handle,          // handle of the iAC indicator
                          int amount               // number of copied values
                          )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iACBuffer array with values from the indicator buffer that ha
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAC indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- now copy the indexes of colors
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy color values from the iAC indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iAD

The function returns the handle of the Accumulation/Distribution indicator. It has only one buffer.

```
int  iAD(
   string               symbol,             // symbol name
   ENUM_TIMEFRAMES      period,             // period
   ENUM_APPLIED_VOLUME  applied_volume      // volume type for calculation
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES enumeration values, 0 means the current timeframe.

applied_volume

[in]   The volume used. Can be any of ENUM_APPLIED_VOLUME values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAD
#property indicator_label1  "iAD"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAD,                // use iAD
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iAD;           // type of the function
input ENUM_APPLIED_VOLUME  volumes;                 // volume used
input string               symbol=" ";              // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;   // timeframe
//--- indicator buffer
double         iADBuffer[];
//--- variable for storing the handle of the iAD indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Accumulation/Distribution indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iADBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iAD)
      handle=iAD(name,period,volumes);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=volumes;
      handle=IndicatorCreate(name,period,IND_AD,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iAD indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Accumulation/Distribution indicator is calculated
   short_name=StringFormat("iAD(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iAD indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iADBuffer array is greater than the number of values in the iAD inc
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iADBuffer array with values of the Accumulation/Distribution indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArrayFromBuffer(iADBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Accumulation/Distribution indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iAD indicator                 |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // indicator buffer of the Accumulation/I
                         int ind_handle,     // handle of the iAD indicator
                         int amount          // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iADBuffer array with values from the indicator buffer that ha
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAD indicator, error code %d",GetLastE
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iADX

The function returns the handle of the Average Directional Movement Index indicator.

```
int  iADX(
   string           symbol,         // symbol name
   ENUM_TIMEFRAMES  period,         // period
   int              adx_period      // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

adx_period

[in] Period to calculate the index.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iADX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```

```
//--- plot ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADX,              // use iADX
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iADX;         // type of the function
input int                  adx_period=14;          // period of calculation
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffers
double         ADXBuffer[];
double         DI_plusBuffer[];
double         DI_minusBuffer[];
//--- variable for storing the handle of the iADX indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average Directional Movement Index ind
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
```

```
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iADX)
      handle=iADX(name,period,adx_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADX,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iADX indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Average Directional Movement Index indicator is ca
   short_name=StringFormat("iADX(%s/%s period=%d)",name,EnumToString(period),adx_peri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//--- number of values copied from the iADX indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means 
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iADXBuffer array is greater than the number of values in the iADX i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the Average Directional Movement Index indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_c
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Average Directional Movement Index indicato
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iADX indicator                |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // indicator buffer of the ADX 
                           double &DIplus_values[],   // indicator buffer for DI+
                           double &DIminus_values[],  // indicator buffer for DI-
                           int ind_handle,            // handle of the iADX indicator
                           int amount                 // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iADXBuffer array with values from the indicator buffer that 
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }

//--- fill a part of the DI_plusBuffer array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }

//--- fill a part of the DI_minusBuffer array with values from the indicator buffer th
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADX indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iADXWilder

The function returns the handle of Average Directional Movement Index by Welles Wilder.

```
int  iADXWilder(
   string           symbol,         // symbol name
   ENUM_TIMEFRAMES  period,         // period
   int              adx_period      // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

adx_period

[in] Period to calculate the index.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                                   iADXWilder.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iADXWilder technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```

```
//--- plot ADX
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADXWilder,        // use iADXWilder
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iADXWilder;   // type of the function
input int                  adx_period=14;          // period of calculation
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffers
double         ADXBuffer[];
double         DI_plusBuffer[];
double         DI_minusBuffer[];
//--- variable for storing the handle of the iADXWilder indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average Directional Movement Index by
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
```

```
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iADXWilder)
      handle=iADXWilder(name,period,adx_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADXW,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iADXWilder indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Average Directional Movement Index by Welles Wilde
   short_name=StringFormat("iADXWilder(%s/%s period=%d)",name,EnumToString(period),adx
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iADXWilder indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iADXBuffer array is greater than the number of values in the iADXW
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the Average Directional Movement Index by Welles 
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Average Directional Movement Index indicat
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+-------------------------------------------------------------------+
//| Filling indicator buffers from the iADXWilder indicator           |
//+-------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // indicator buffer of the ADX
                           double &DIplus_values[],   // indicator buffer for DI+
                           double &DIminus_values[],  // indicator buffer for DI-
                           int ind_handle,            // handle of the iADXWilder ind
                           int amount                 // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iADXBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }

//--- fill a part of the DI_plusBuffer array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }

//--- fill a part of the DI_plusBuffer array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iADXWilder indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iAlligator

The function returns the handle of the Alligator indicator.

```
int  iAlligator(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 jaw_period,        // period for the calculation of jaws
   int                 jaw_shift,         // horizontal shift of jaws
   int                 teeth_period,      // period for the calculation of teeth
   int                 teeth_shift,       // horizontal shift of teeth
   int                 lips_period,       // period for the calculation of lips
   int                 lips_shift,        // horizontal shift of lips
   ENUM_MA_METHOD      ma_method,         // type of smoothing
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

jaw_period

[in] Averaging period for the blue line (Alligator's Jaw)

jaw_shift

[in] The shift of the blue line relative to the price chart.

teeth_period

[in] Averaging period for the red line (Alligator's Teeth).

teeth_shift

[in] The shift of the red line relative to the price chart.

lips_period

[in] Averaging period for the green line (Alligator's lips).

lips_shift

[in] The shift of the green line relative to the price chart.

ma_method

[in] The method of averaging. Can be any of the ENUM_MA_METHOD values.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - GATORJAW_LINE, 1 - GATORTEETH_LINE, 2 - GATORLIPS_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iAlligator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAlligator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Alligator.

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- plot Jaws
#property indicator_label1  "Jaws"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot Teeth
#property indicator_label2  "Teeth"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot Lips
#property indicator_label3  "Lips"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrLime
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAlligator,        // use iAlligator
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iAlligator;   // type of the function
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
input int                  jaw_period=13;          // period of the Jaw line
input int                  jaw_shift=8;            // shift of the Jaw line
input int                  teeth_period=8;         // period of the Teeth line
input int                  teeth_shift=5;          // shift of the Teeth line
input int                  lips_period=5;          // period of the Lips line
input int                  lips_shift=3;           // shift of the Lips line
input ENUM_MA_METHOD       MA_method=MODE_SMMA;    // method of averaging of the Allig
input ENUM_APPLIED_PRICE   applied_price=PRICE_MEDIAN;// type of price used for calcul
//--- indicator buffers
double         JawsBuffer[];
double         TeethBuffer[];
double         LipsBuffer[];
//--- variable for storing the handle of the iAlligator indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Alligator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,JawsBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,TeethBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LipsBuffer,INDICATOR_DATA);
//--- set shift of each line
   PlotIndexSetInteger(0,PLOT_SHIFT,jaw_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,teeth_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,lips_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
```

```
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iAlligator)
      handle=iAlligator(name,period,jaw_period,jaw_shift,teeth_period,
                        teeth_shift,lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[8];
     //--- periods and shifts of the Alligator lines
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
//--- type of smoothing
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
//--- type of price
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
//--- create handle
      handle=IndicatorCreate(name,period,IND_ALLIGATOR,8,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iAlligator indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Alligator indicator is calculated for
   short_name=StringFormat("iAlligator(%s/%s, %d,%d,%d,%d,%d,%d)",name,EnumToString(pe
                             jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,l
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iAlligator indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the JawsBuffer array is greater than the number of values in the iAllig
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the Alligator indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(JawsBuffer,jaw_shift,TeethBuffer,teeth_shift,LipsBuffer,l
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Alligator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iAlligator indicator          |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &jaws_buffer[],  // indicator buffer for the Jaw li
                           int j_shift,            // shift of the Jaw line
                           double &teeth_buffer[], // indicator buffer for the Teeth
                           int t_shift,            // shift of the Teeth line
                           double &lips_buffer[],  // indicator buffer for the Lips l
                           int l_shift,            // shift of the Lips line
                           int ind_handle,         // handle of the iAlligator indicat
                           int amount              // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the JawsBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-j_shift,amount,jaws_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }

//--- fill a part of the TeethBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,-t_shift,amount,teeth_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }

//--- fill a part of the LipsBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,2,-l_shift,amount,lips_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAlligator indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iAMA

The function returns the handle of the Adaptive Moving Average indicator. It has only one buffer.

```
int  iAMA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ama_period,        // average period for AMA
   int                 fast_ma_period,    // fast MA period
   int                 slow_ma_period,    // slow MA period
   int                 ama_shift,         // horizontal shift of the indicator
   ENUM_APPLIED_PRICE  applied_price      // type of the price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ama_period

[in] The calculation period, on which the efficiency coefficient is calculated.

fast_ma_period

[in] Fast period for the smoothing coefficient calculation for a rapid market.

slow_ma_period

[in] Slow period for the smoothing coefficient calculation in the absence of trend.

ama_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard AMA."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAMA
#property indicator_label1  "iAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAMA,              // use iAMA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iAMA;          // type of the function
input string               symbol=" ";              // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;   // timeframe
input int                  ama_period=15;           // period of calculation
input int                  fast_ma_period=2;        // period of fast MA
input int                  slow_ma_period=30;       // period of slow MA
input int                  ama_shift=0;             // horizontal shift
input ENUM_APPLIED_PRICE   applied_price;           // type of price
//--- indicator buffer
double         iAMABuffer[];
//--- variable for storing the handle of the iAMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Adaptive Moving Average indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,iAMABuffer,INDICATOR_DATA);
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ama_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iAMA)
      handle=iAMA(name,period,ama_period,fast_ma_period,slow_ma_period,ama_shift,appl
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[5];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ama_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=fast_ma_period;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=slow_ma_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ama_shift;
      //--- type of price
      pars[4].type=TYPE_INT;
      pars[4].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_AMA,5,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iAMA indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Adaptive Moving Average indicator is calculated fo
```

```
   short_name=StringFormat("iAMA(%s/%s,%d,%d,%d,d)",name,EnumToString(period),ama_per
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iAMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iAMABuffer array is greater than the number of values in the iAMA
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the Adaptive Moving Average indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArrayFromBuffer(iAMABuffer,ama_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Adaptive Moving Average indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffer from the iAMA indicator                 |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  // indicator buffer of the AMA line
                         int a_shift,           // shift of the AMA line
                         int ind_handle,        // handle of the iAMA indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iAMABuffer array with values from the indicator buffer that 
   if(CopyBuffer(ind_handle,0,-a_shift,amount,ama_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAMA indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iAO

The function returns the handle of the Awesome Oscillator indicator. It has only one buffer.

```
int  iAO(
   string           symbol,      // symbol name
   ENUM_TIMEFRAMES  period       // period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                      Demo_iAO.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iAO technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- the iAO plot
#property indicator_label1  "iAO"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen,clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAO,               // use iAO
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iAO;          // type of the function
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffers
double         iAOBuffer[];
double         iAOColors[];
//--- variable for storing the handle of the iAO indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Awesome Oscillator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,iAOBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iAOColors,INDICATOR_COLOR_INDEX);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iAO)
      handle=iAO(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AO);
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iAO indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Awesome Oscillator indicator is calculated for
   short_name=StringFormat("iAO(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iAO indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iAOBuffer array is greater than the number of values in the iAO in
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iAOBuffer and iAOColors arrays with values from the Awesome Oscillator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready ye
   if(!FillArraysFromBuffer(iAOBuffer,iAOColors,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Awesome Oscillator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iAO indicator                 |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],        // indicator buffer of Awesome Osc
                          double &color_indexes[], // color buffer (for storing of co
                          int ind_handle,          // handle of the iAO indicator
                          int amount               // number of copied values
                          )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iAOBuffer array with values from the indicator buffer that ha
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iAO indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- now copy the indexes of colors
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy color values from the iAO indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iATR

The function returns the handle of the Average True Range indicator. It has only one buffer.

```
int  iATR(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   int              ma_period      // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] The value of the averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iATR.mq5 |
//|                         Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iATR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iATR
#property indicator_label1  "iATR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iATR,// use iATR
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input int                  atr_period=14;          // period of calculation
input Creation             type=Call_iATR;         // type of the function
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double         iATRBuffer[];
//--- variable for storing the handle of the iAC indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Average True Range indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iATRBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iATR)
      handle=iATR(name,period,atr_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=atr_period;
      handle=IndicatorCreate(name,period,IND_ATR,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iATR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Average True Range indicator is calculated for
   short_name=StringFormat("iATR(%s/%s, period=%d)",name,EnumToString(period),atr_per
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iATR indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iATRBuffer array is greater than the number of values in the iATR
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iATRBuffer array with values of the Average True Range indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iATRBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Average True Range indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iATR indicator                 |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer for ATR values
                         int ind_handle,    // handle of the iATR indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iATRBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iATR indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iBearsPower

The function returns the handle of the Bears Power indicator. It has only one buffer.

```
int  iBearsPower(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] The value of the averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iBearsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBearsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iBearsPower plot
#property indicator_label1  "iBearsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBearsPower,       // use iBearsPower
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iBearsPower;  // type of the function
input int                  ma_period=13;           // period of moving average
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double         iBearsPowerBuffer[];
//--- variable for storing the handle of the iBearsPower indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bears Power indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iBearsPowerBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iBearsPower)
      handle=iBearsPower(name,period,ma_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- period of ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BEARS,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iBearsPower indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Bears Power indicator is calculated for
   short_name=StringFormat("iBearsPower(%s/%s, period=%d)",name,EnumToString(period),r
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iBearsPower indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iBearsPowerBuffer array is greater than the number of values in the
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iBearsPowerBuffer array with values of the Bears Power indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iBearsPowerBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Bears Power indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iBearsPower indicator         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer for Bears Power values
                         int ind_handle,    // handle of the iBearsPower indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iBearsPowerBuffer array with values from the indicator buffe
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBearsPower indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iBands

The function returns the handle of the Bollinger Bands® indicator.

```
int  iBands(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 bands_period,      // period for average line calculation
   int                 bands_shift,       // horizontal shift of the indicator
   double              deviation,         // number of standard deviations
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

bands_period

[in] The averaging period of the main line of the indicator.

bands_shift

[in] The shift the indicator relative to the price chart.

deviation

[in] Deviation from the main line.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - BASE_LINE, 1 - UPPER_BAND, 2 - LOWER_BAND

Example:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iBands.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBands technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- the Upper plot
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Lower plot
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrMediumSeaGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- the Middle plot
#property indicator_label3  "Middle"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrMediumSeaGreen
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBands,             // use iBands
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iBands;          // type of the function
input int                  bands_period=20;           // period of moving average
input int                  bands_shift=0;             // shift
input double               deviation=2.0;             // number of standard deviations
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffers
double         UpperBuffer[];
double         LowerBuffer[];
double         MiddleBuffer[];
//--- variable for storing the handle of the iBands indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bollinger Bands indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,MiddleBuffer,INDICATOR_DATA);
//--- set shift of each line
   PlotIndexSetInteger(0,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,bands_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iBands)
      handle=iBands(name,period,bands_period,bands_shift,deviation,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period of ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=bands_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=bands_shift;
      //--- number of standard deviation
      pars[2].type=TYPE_DOUBLE;
      pars[2].double_value=deviation;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_BANDS,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iBands indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Bollinger Bands indicator is calculated for
   short_name=StringFormat("iBands(%s/%s, %d,%d,%G,%s)",name,EnumToString(period),
                           bands_period,bands_shift,deviation,EnumToString(applied_pr
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iBands indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the size of indicator buffers is greater than the number of values in t
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the Bollinger Bands indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(MiddleBuffer,UpperBuffer,LowerBuffer,bands_shift,handle,v
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Bollinger Bands indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iBands indicator              |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &base_values[],     // indicator buffer of the middl
                           double &upper_values[],    // indicator buffer of the uppe
                           double &lower_values[],    // indicator buffer of the lowe
                           int shift,                 // shift
                           int ind_handle,            // handle of the iBands indicat
                           int amount                 // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the MiddleBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-shift,amount,base_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
```

```
//--- fill a part of the UpperBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,-shift,amount,upper_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the LowerBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,2,-shift,amount,lower_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBands indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iBullsPower

The function returns the handle of the Bulls Power indicator. It has only one buffer.

```
int  iBullsPower(
   string              symbol,           // symbol name
   ENUM_TIMEFRAMES     period,           // period
   int                 ma_period,        // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] The averaging period for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                            Demo_iBullsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBullsPower technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iBullsPower plot
#property indicator_label1  "iBullsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBullsPower,       // use iBullsPower
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iBullsPower;  // type of the function
input int                  ma_period=13;           // period of moving average
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
//--- indicator buffer
double         iBullsPowerBuffer[];
//--- variable for storing the handle of the iBullsPower indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Bulls Power indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iBullsPowerBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iBullsPower)
      handle=iBullsPower(name,period,ma_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- period of ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BULLS,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iBullsPower indicator for the symbo
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Bulls Power indicator is calculated for
   short_name=StringFormat("iBullsPower(%s/%s, period=%d)",name,EnumToString(period),
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iBullsPower indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iBullsPowerBuffer array is greater than the number of values in the
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iBullsPowerBuffer array with values of the Bulls Power indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iBullsPowerBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Bulls Power indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iBullsPower indicator         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Bulls Power values
                         int ind_handle,    // handle of the iBullsPower indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iBullsPowerBuffer array with values from the indicator buffe
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBullsPower indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
//+------------------------------------------------------------------+
```

iCCI

The function returns the handle of the Commodity Channel Index indicator. It has only one buffer.

```
int  iCCI(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] The averaging period for the indicator calculation.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iCCI.mq5 |
//|                         Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iCCI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
```

```
#property indicator_plots   1
//--- the iCCI plot
#property indicator_label1  "iCCI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- horizontal levels in the indicator window
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iCCI,              // use iCCI
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iCCI;              // type of the function
input int                  ma_period=14;                // period of moving average
input ENUM_APPLIED_PRICE   applied_price=PRICE_TYPICAL; // type of price
input string               symbol=" ";                  // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- indicator buffer
double         iCCIBuffer[];
//--- variable for storing the handle of the iCCI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Commodity Channel Index indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iCCIBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iCCI)
      handle=iCCI(name,period,ma_period,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- type of price
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_CCI,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iCCI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the CCI indicator is calculated for
   short_name=StringFormat("iCCI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iCCI indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iCCIBuffer array is greater than the number of values in the iCCI
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iCCIBuffer array with values of the Commodity Channel Index indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iCCIBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Commodity Channel Index indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iCCI indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Commodity Channel 
                         int ind_handle,    // handle of the iCCI indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
```

```
   ResetLastError();
//--- fill a part of the iCCIBuffer array with values from the indicator buffer that 
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iCCI indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iChaikin

The function returns the handle of the Chaikin Oscillator indicator. It has only one buffer.

```
int  iChaikin(
   string               symbol,             // symbol name
   ENUM_TIMEFRAMES      period,             // period
   int                  fast_ma_period,     // fast period
   int                  slow_ma_period,     // slow period
   ENUM_MA_METHOD       ma_method,          // smoothing type
   ENUM_APPLIED_VOLUME  applied_volume      // type of volume
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

fast_ma_period

[in] Fast averaging period for calculations.

slow_ma_period

[in] Slow averaging period for calculations.

ma_method

[in] Smoothing type. Can be one of the averaging constants of ENUM_MA_METHOD.

applied_volume

[in] The volume used. Can be one of the constants of ENUM_APPLIED_VOLUME.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                Demo_iChaikin.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```

```
#property description "of indicator buffers for the iChaikin technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iChaikin plot
#property indicator_label1  "iChaikin"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iChaikin,          // use iChaikin
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iChaikin;           // type of the function
input int                  fast_ma_period=3;             // period of fast ma
input int                  slow_ma_period=10;            // period of slow ma
input ENUM_MA_METHOD       ma_method=MODE_EMA;           // type of smoothing
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // type of volume
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffer
double         iChaikinBuffer[];
//--- variable for storing the handle of the iChaikin indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Chaikin Oscillator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iChaikinBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
```

```
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iChaikin)
      handle=iChaikin(name,period,fast_ma_period,slow_ma_period,ma_method,applied_volu
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period of fast ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ma_period;
      //--- period of slow ma
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ma_period;
      //--- type of smoothing
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- type of volume
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_CHAIKIN,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iChaikin indicator for the symbol %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Chaikin Oscillator indicator is calculated for
   short_name=StringFormat("iChaikin(%s/%s, %d, %d, %s, %s)",name,EnumToString(period)
                           fast_ma_period,slow_ma_period,
                           EnumToString(ma_method),EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iChaikin indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iChaikinBuffer array is greater than the number of values in the i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iChaikinBuffer array with values of the Chaikin Oscillator indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet
   if(!FillArrayFromBuffer(iChaikinBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Chaikin Oscillator indicator
```

```
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iChaikin indicator            |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Chaikin Oscillator
                         int ind_handle,    // handle of the iChaikin indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iChaikinBuffer array with values from the indicator buffer th
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iChaikin indicator, error code %d",Get
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iCustom

The function returns the handle of a specified custom indicator.

```
int  iCustom(
   string           symbol,     // symbol name
   ENUM_TIMEFRAMES  period,     // period
   string           name        // folder/custom_indicator_name
   ...                          // list of indicator input parameters
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

name

[in] Custom indicator name. If the name starts with the reverse slash '\', the EX5 indicator file is searched for relative to the MQL5 root directory. So, for a call of iCustom(Symbol(), Period(), "\FirstIndicator"...), the indicator will be loaded as MQL5\FirstIndicator.ex5. If the file is not found at this path, error 4802 (ERR_INDICATOR_CANNOT_CREATE) is returned.

If the path does not start with '\', the indicator is searched and downloaded as follows:

- First, the indicator EX5 file is searched for in the folder where the EX5 file of the calling program is located. For example, the CrossMA.EX5 EA is located in MQL5\Experts\MyExperts and contains the iCustom call (Symbol(), Period(), "SecondIndicator"...). In this case, the indicator is searched for in MQL5\Experts\MyExperts\SecondIndicator.ex5.
- If the indicator is not found in the same directory, the search is performed relative to the MQL5\Indicators indicator root directory. In other words, the search for the MQL5\Indicators\SecondIndicator.ex5 file is performed. If the indicator is still not found, the function returns INVALID_HANDLE and the error 4802 (ERR_INDICATOR_CANNOT_CREATE) is triggered.

If the path to the indicator is set in the subdirectory (for example, MyIndicators\ThirdIndicator), the search is first performed in the calling program folder (the EA is located in MQL5\Experts\MyExperts) in MQL5\Experts\MyExperts\MyIndicators\ThirdIndicator.ex5. If unsuccessful, the search for the MQL5\Indicators\MyIndicators\ThirdIndicator.ex5 file is performed. Make sure to use the double reverse slash '\\' as a separator in the path, for example iCustom(Symbol(), Period(), "MyIndicators\\ThirdIndicator"...)

...

[in] input-parameters of a custom indicator, separated by commas. Type and order of parameters must match. If there is no parameters specified, then default values will be used.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

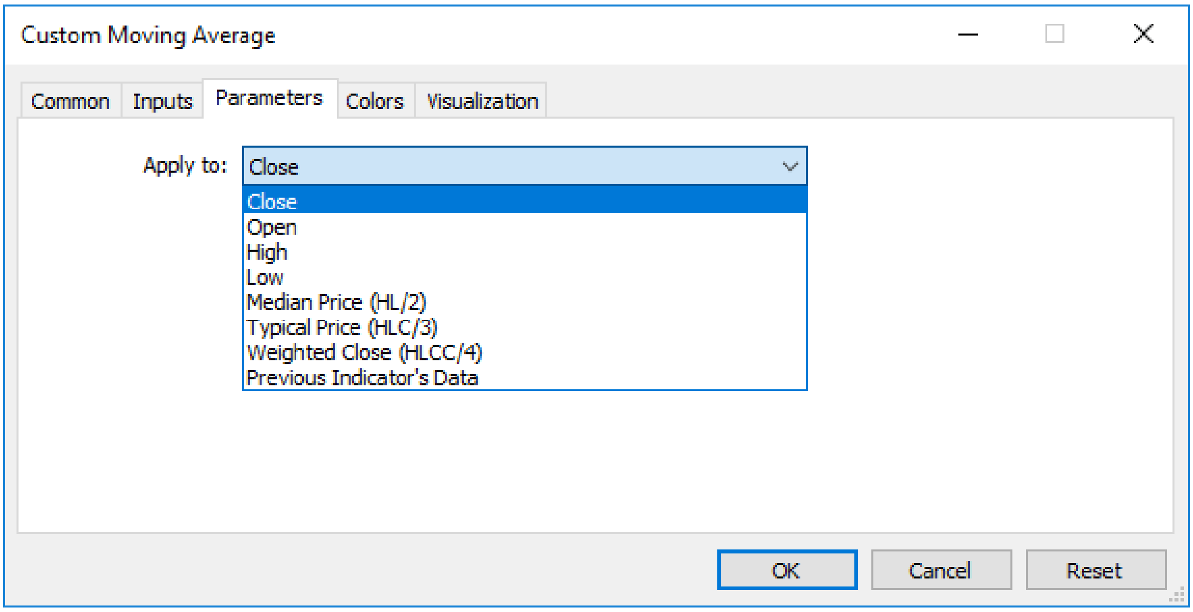
Note

A custom indicator must be compiled (with extension EX5) and located in the directory MQL5/Indicators of the client terminal or its subdirectory.

Indicators that require testing are defined automatically from the call of the iCustom() function, if the corresponding parameter is set through a constant string. For all other cases (use of the IndicatorCreate() function or use of a non-constant string in the parameter that sets the indicator name) the property #property tester_indicator is required:

```
#property tester_indicator "indicator_name.ex5"
```

If the first call form is used in the indicator, then at the custom indicator start you can additionally indicate data for calculation in its "Parameters" tab. If the "Apply to" parameter is not selected explicitly, the default calculation is based on the values of "Close" prices.


Custom Moving Average
Common
Inputs
Parameters
Colors
Visualization
Apply to:
Close
Close
Open
High
Low
Median Price (HL/2)
Typical Price (HLC/3)
Weighted Close (HLCC/4)
Previous Indicator's Data
OK
Cancel
Reset

When you call a custom indicator from mql5-program, the Applied_Price parameter or a handle of another indicator should be passed last, after all input variables of the custom indicator.

See also

Program Properties, Timeseries and Indicators Access,IndicatorCreate(), IndicatorRelease()

Example:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int MA_Period=21;
input int MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMA;
//--- indicator buffers
double         Label1Buffer[];
//--- Handle of the Custom Moving Average.mq5 custom indicator
int MA_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
   ResetLastError();
   MA_handle=iCustom(NULL,0,"Examples\\Custom Moving Average",
                     MA_Period,
                     MA_Shift,
                     MA_Method,
                     PRICE_CLOSE // using the close prices
                     );
   Print("MA_handle = ",MA_handle,"  error = ",GetLastError());
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- Copy the values of the indicator Custom Moving Average to our indicator buffer
   int copy=CopyBuffer(MA_handle,0,0,rates_total,Label1Buffer);
   Print("copy = ",copy,"    rates_total = ",rates_total);
//--- If our attempt has failed - Report this
   if(copy<=0)
      Print("An attempt to get the values if Custom Moving Average has failed");
//--- return value of prev_calculated for next call
```

```
    return(rates_total);
  }
//+------------------------------------------------------------------+
```

iDEMA

The function returns the handle of the Double Exponential Moving Average indicator. It has only one buffer.

```
int  iDEMA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   int                 ma_shift,          // horizontal shift
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

ma_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iDEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```

```
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iDEMA plot
#property indicator_label1  "iDEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDEMA,             // use iDEMA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iDEMA;           // type of te function
input int                  ma_period=14;              // period of moving average
input int                  ma_shift=0;                // shift
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iDEMABuffer[];
//--- variable for storing the handle of the iDEMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Double Exponential Moving Average indic
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iDEMABuffer,INDICATOR_DATA);
   //--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
```

```
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iDEMA)
      handle=iDEMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[3];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of price
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_DEMA,3,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iDEMA indicator for the symbol %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Double Exponential Moving Average indicator is cal
   short_name=StringFormat("iDEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iDEMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iDEMABuffer array is greater than the number of values in the iDEM
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iDEMABuffer array with values of the Double Exponential Moving Average
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iDEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Double Exponential Moving Average indicato
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iDEMA indicator               |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Double Exponential
                         int shift,         // shift
                         int ind_handle,    // handle of the iDEMA indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iDEMABuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iDEMA indicator, error code %d",GetLas
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iDeMarker

The function returns the handle of the DeMarker indicator. It has only one buffer.

```
int  iDeMarker(
   string           symbol,         // symbol name
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period       // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                               Demo_iDeMarker.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iDeMarker technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iDeMarker plot
#property indicator_label1  "iDeMarker"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- horizontal levels in the indicator window
#property indicator_level1  0.3
#property indicator_level2  0.7
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDeMarker,          // use iDeMarker
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iDeMarker;         // type of the function
input int                  ma_period=14;                // period of moving average
input string               symbol=" ";                  // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- indicator buffer
double         iDeMarkerBuffer[];
//--- variable for storing the handle of the iDeMarker indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the DeMarker indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iDeMarkerBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iDeMarker)
      handle=iDeMarker(name,period,ma_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_DEMARKER,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iDeMarker indicator for the symbol 
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the DeMarker indicator is calculated for
   short_name=StringFormat("iDeMarker(%s/%s, period=%d)",name,EnumToString(period),ma_
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iDeMarker indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
```

```
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iDeMarkerBuffer array is greater than the number of values in the 
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iDeMarkerBuffer array with values of the DeMarker indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iDeMarkerBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the DeMarker indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iDeMarker indicator           |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer for DeMarker values
                         int ind_handle,    // handle of the iDeMarker indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iDeMarkerBuffer array with values from the indicator buffer t
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iDeMarker indicator, error code %d",Ge
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iEnvelopes

The function returns the handle of the Envelopes indicator.

```
int  iEnvelopes(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // period for the average line calculation
   int                 ma_shift,          // horizontal shift of the indicator
   ENUM_MA_METHOD      ma_method,         // type of smoothing
   ENUM_APPLIED_PRICE  applied_price,     // type of price or handle
   double              deviation          // deviation of boundaries from the midline
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the main line.

ma_shift

[in] The shift of the indicator relative to the price chart.

ma_method

[in] Smoothing type. Can be one of the values of ENUM_MA_METHOD.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

deviation

[in] The deviation from the main line (in percents).

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - UPPER_LINE, 1 - LOWER_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iEnvelopes.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iEnvelopes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- the Upper plot
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Lower plot
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iEnvelopes,        // use iEnvelopes
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iEnvelopes;      // type of the function
input int                  ma_period=14;              // period of moving average
input int                  ma_shift=0;                // shift
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // type of smoothing
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input double               deviation=0.1;             // deviation of borders from the
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         UpperBuffer[];
double         LowerBuffer[];
//--- variable for storing the handle of the iEnvelopes indicator
```

```
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Envelopes indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
//--- set shift of each line
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iEnvelopes)
      handle=iEnvelopes(name,period,ma_period,ma_shift,ma_method,applied_price,deviat
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[5];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of smoothing
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      //--- type of price
      pars[4].type=TYPE_DOUBLE;
      pars[4].double_value=deviation;
      handle=IndicatorCreate(name,period,IND_ENVELOPES,5,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iEnvelopes indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Envelopes indicator is calculated for
   short_name=StringFormat("iEnvelopes(%s/%s, %d, %d, %s,%s, %G)",name,EnumToString(pe
   ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price),deviation);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iEnvelopes indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the UpperBuffer array is greater than the number of values in the iEnve
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the UpperBuffer and LowerBuffer arrays with values from the Envelopes indic
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArraysFromBuffers(UpperBuffer,LowerBuffer,ma_shift,handle,values_to_copy))
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Envelopes indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iEnvelopes indicator          |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &upper_values[],    // indicator buffer of the upper
                           double &lower_values[],    // indicator of the lower border
                           int shift,                 // shift
                           int ind_handle,            // handle of the iEnvelopes indi
                           int amount                 // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the UpperBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-shift,amount,upper_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iEnvelopes indicator, error code %d",G
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- fill a part of the LowerBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,-shift,amount,lower_values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iEnvelopes indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iForce

The function returns the handle of the Force Index indicator. It has only one buffer.

```
int  iForce(
   string               symbol,            // symbol name
   ENUM_TIMEFRAMES      period,            // period
   int                  ma_period,         // averaging period
   ENUM_MA_METHOD       ma_method,         // smoothing type
   ENUM_APPLIED_VOLUME  applied_volume     // volume type for calculation
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the indicator calculations.

ma_method

[in] Smoothing type. Can be one of the values of ENUM_MA_METHOD.

applied_volume

[in] The volume used. Can be one of the values of ENUM_APPLIED_VOLUME.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iForce.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iForce technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
```

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- drawing iForce
#property indicator_label1  "iForce"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iForce,            // use iForce
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iForce;             // type of the function
input int                  ma_period=13;                 // period of averaging
input ENUM_MA_METHOD       ma_method=MODE_SMA;           // type of smoothing
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // type of volume
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffer
double         iForceBuffer[];
//--- variable for storing the handle of the iForce indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Force indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iForceBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
```

```
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iForce)
      handle=iForce(name,period,ma_period,ma_method,applied_volume);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[3];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- type of smoothing
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_method;
      //--- type of volume
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_volume;
      //--- type of price
      handle=IndicatorCreate(name,period,IND_FORCE,3,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iForce indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Force indicator is calculated for
   short_name=StringFormat("iForce(%s/%s, %d, %s, %s)",name,EnumToString(period),
                           ma_period,EnumToString(ma_method),EnumToString(applied_vol
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//--- number of values copied from the iForce indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means 
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iForceBuffer array is greater than the number of values in the iFor
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iForceBuffer array with values of the Force indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iForceBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Force indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iForce indicator              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Force Index values
                         int ind_handle,     // handle of the iForce indicator
                         int amount          // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iForceBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iForce indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iFractals

The function returns the handle of the Fractals indicator.

```
int  iFractals(
   string           symbol,      // symbol name
   ENUM_TIMEFRAMES  period       // period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - UPPER_LINE, 1 - LOWER_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iFractals.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFractals technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- the FractalUp plot
```

```
#property indicator_label1  "FractalUp"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
//--- the FractalDown plot
#property indicator_label2  "FractalDown"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFractals,         // use iFractals
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iFractals;          // type of the function
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffers
double         FractalUpBuffer[];
double         FractalDownBuffer[];
//--- variable for storing the handle of the iFractals indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Fractals indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,FractalUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,FractalDownBuffer,INDICATOR_DATA);
//--- set codes using a symbol from the Wingdings charset for the PLOT_ARROW property
   PlotIndexSetInteger(0,PLOT_ARROW,217); // arrow up
   PlotIndexSetInteger(1,PLOT_ARROW,218); // arrow down
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iFractals)
      handle=iFractals(name,period);
   else
      handle=IndicatorCreate(name,period,IND_FRACTALS);
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iFractals indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Fractals indicator is calculated for
   short_name=StringFormat("iFractals(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iFractals indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
```

```
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the FractalUpBuffer array is greater than the number of values in the
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the FractalUpBuffer and FractalDownBuffer arrays with values from the Fract
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArraysFromBuffers(FractalUpBuffer,FractalDownBuffer,handle,values_to_copy))
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Fractals indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iFractals indicator           |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &up_arrows[],        // indicator buffer for up arr
                           double &down_arrows[],      // indicator buffer for down a
                           int ind_handle,             // handle of the iFractals ind
                           int amount                  // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the FractalUpBuffer array with values from the indicator buffer
   if(CopyBuffer(ind_handle,0,0,amount,up_arrows)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iFractals indicator to the FractalUpBu
                  GetLastError());
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- fill a part of the FractalDownBuffer array with values from the indicator buffe
   if(CopyBuffer(ind_handle,1,0,amount,down_arrows)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iFractals indicator to the FractalDow
                  GetLastError());
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iFrAMA

The function returns the handle of the Fractal Adaptive Moving Average indicator. It has only one buffer.

```
int  iFrAMA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   int                 ma_shift,          // horizontal shift on the chart
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Period (bars count) for the indicator calculations.

ma_shift

[in] Shift of the indicator in the price chart.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iFrAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iFrAMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```

```
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- drawing iFrAMA
#property indicator_label1  "iFrAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFrAMA,            // use iFrAMA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iFrAMA;              // type of the function
input int                  ma_period=14;                  // period of averaging
input int                  ma_shift=0;                    // shift
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE;     // type of price
input string               symbol=" ";                    // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;         // timeframe
//--- indicator buffer
double         iFrAMABuffer[];
//--- variable for storing the handle of the iFrAMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Fractal Adaptive Moving Average indicat
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iFrAMABuffer,INDICATOR_DATA);
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
```

```
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iFrAMA)
      handle=iFrAMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[3];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of price
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      //--- type of price
      handle=IndicatorCreate(name,period,IND_FRAMA,3,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iFrAMA indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the iFrAMA indicator is calculated for
   short_name=StringFormat("iFrAMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iFrAMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means 
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iFrAMABuffer array is greater than the number of values in the iFrA
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iFrAMABuffer array with values of the Fractal Adaptive Moving Average i
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iFrAMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Fractal Adaptive Moving Average indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iFrAMA indicator              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Fractal Adaptive Mo
                         int shift,          // shift
                         int ind_handle,     // handle of the iFrAMA indicator
                         int amount          // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iFrAMABuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iFrAMA indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iGator

The function returns the handle of the Gator indicator. The Oscillator shows the difference between the blue and red lines of Alligator (upper histogram) and difference between red and green lines (lower histogram).

```
int  iGator(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 jaw_period,        // period for the calculation of the jaws
   int                 jaw_shift,         // jaws horizontal shift
   int                 teeth_period,      // period for the calculation of the teeth
   int                 teeth_shift,       // teeth horizontal shift
   int                 lips_period,       // period for the calculation of the lips
   int                 lips_shift,        // lips horizontal shift
   ENUM_MA_METHOD      ma_method,         // type of smoothing
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

jaw_period

[in] Averaging period for the blue line (Alligator's Jaw).

jaw_shift

[in] The shift of the blue line relative to the price chart. It isn't directly connected with the visual shift of the indicator histogram.

teeth_period

[in] Averaging period for the red line (Alligator's Teeth).

teeth_shift

[in] The shift of the red line relative to the price chart. It isn't directly connected with the visual shift of the indicator histogram.

lips_period

[in] Averaging period for the green line (Alligator's lips).

lips_shift

[in] The shift of the green line relative to the price charts. It isn't directly connected with the visual shift of the indicator histogram.

ma_method

[in] Smoothing type. Can be one of the values of ENUM_MA_METHOD.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

Buffer numbers: 0 - UPPER_HISTOGRAM, 1 - color buffer of the upper histogram, 2 - LOWER_HISTOGRAM, 3 - color buffer of the lower histogram.

Example:

```
//+------------------------------------------------------------------+
//|                                                Demo_iGator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iGator technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters are as in a standard Gator Oscillator."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
//--- drawing GatorUp
#property indicator_label1  "GatorUp"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- drawing GatorDown
#property indicator_label2  "GatorDown"
#property indicator_type2   DRAW_COLOR_HISTOGRAM
#property indicator_color2  clrGreen, clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
```

```
  {
   Call_iGator,            // use iGator
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iGator;       // type of the function
input string               symbol=" ";             // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // timeframe
input int                  jaw_period=13;          // period of the Jaw line
input int                  jaw_shift=8;            // shift of the Jaw line
input int                  teeth_period=8;         // period of the Teeth line
input int                  teeth_shift=5;          // shift of the Teeth line
input int                  lips_period=5;          // period of the Lips line
input int                  lips_shift=3;           // shift of the Lips line
input ENUM_MA_METHOD       MA_method=MODE_SMMA;    // method of averaging of the Alli
input ENUM_APPLIED_PRICE   applied_price=PRICE_MEDIAN;// type of price used for calcul
//--- indicator buffers
double         GatorUpBuffer[];
double         GatorUpColors[];
double         GatorDownBuffer[];
double         GatorDownColors[];
//--- variable for storing the handle of the iGator indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- shift values for the upper and lower histograms
int shift;
//--- we will keep the number of values in the Gator Oscillator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,GatorUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,GatorUpColors,INDICATOR_COLOR_INDEX);
   SetIndexBuffer(2,GatorDownBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,GatorDownColors,INDICATOR_COLOR_INDEX);
/*
  All the shifts specified in the parameters refer to the Alligator indicator on the
  That's is why they don't move the Gator indicator itself, but they move the Alligat
  which values are used for calculation of the Gator Oscillator!
*/
//--- let's calculate the shift for the upper and lower histograms, that is equal to
   shift=MathMin(jaw_shift,teeth_shift);
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
```

```
//--- despite the indicator contains two histograms, the same shift is used - this is
   PlotIndexSetInteger(1,PLOT_SHIFT,shift);

//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iGator)
      handle=iGator(name,period,jaw_period,jaw_shift,teeth_period,teeth_shift,
                    lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[8];
      //--- periods and shifts of the Alligator lines
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
      //--- type of smoothing
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
      //--- type of price
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
      //--- create handle
      handle=IndicatorCreate(name,period,IND_GATOR,8,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iGator indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Gator Oscillator indicator is calculated for
   short_name=StringFormat("iGator(%s/%s, %d, %d ,%d, %d, %d, %d)",name,EnumToString(
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iGator indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the GatorUpBuffer array is greater than the number of values in the iGa
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the Gator Oscillator indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(GatorUpBuffer,GatorUpColors,GatorDownBuffer,GatorDownColo
      shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Gator Oscillator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iGator indicator              |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &ups_buffer[],         // indicator buffer for the u
                           double &up_color_buffer[],    // indicator buffer for price
                           double &downs_buffer[],       // indicator buffer for the l
                           double &downs_color_buffer[], // indicator buffer for price
                           int u_shift,                  // shift for the upper and lo
                           int ind_handle,               // handle of the iGator indic
                           int amount                    // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the GatorUpBuffer array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,0,-u_shift,amount,ups_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the GatorUpColors array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,1,-u_shift,amount,up_color_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
```

```
//--- fill a part of the GatorDownBuffer array with values from the indicator buffer t
   if(CopyBuffer(ind_handle,2,-u_shift,amount,downs_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the GatorDownColors array with values from the indicator buffer t
   if(CopyBuffer(ind_handle,3,-u_shift,amount,downs_color_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iGator indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iIchimoku

The function returns the handle of the Ichimoku Kinko Hyo indicator.

```
int  iIchimoku(
   string           symbol,           // symbol name
   ENUM_TIMEFRAMES  period,           // period
   int              tenkan_sen,       // period of Tenkan-sen
   int              kijun_sen,        // period of Kijun-sen
   int              senkou_span_b     // period of Senkou Span B
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

tenkan_sen

[in] Averaging period for Tenkan Sen.

kijun_sen

[in] Averaging period for Kijun Sen.

senkou_span_b

[in] Averaging period for Senkou Span B.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - TENKANSEN_LINE, 1 - KIJUNSEN_LINE, 2 - SENKOUSPANA_LINE, 3 - SENKOUSPANB_LINE, 4 - CHIKOUSPAN_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iIchimoku.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
```

```
#property description "of indicator buffers for the iIchimoku technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters just like in the standard Ichimoku Kinko 

#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   4
//--- the Tenkan_sen plot
#property indicator_label1  "Tenkan_sen"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Kijun_sen plot
#property indicator_label2  "Kijun_sen"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrBlue
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- the Senkou_Span plot
#property indicator_label3  "Senkou Span A;Senkou Span B" // two fields will be shown
#property indicator_type3   DRAW_FILLING
#property indicator_color3  clrSandyBrown, clrThistle
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//--- the Chikou_Span plot
#property indicator_label4  "Chinkou_Span"
#property indicator_type4   DRAW_LINE
#property indicator_color4  clrLime
#property indicator_style4  STYLE_SOLID
#property indicator_width4  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iIchimoku,          // use iIchimoku
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iIchimoku;       // type of the function
input int                  tenkan_sen=9;              // period of Tenkan-sen
input int                  kijun_sen=26;              // period of Kijun-sen
input int                  senkou_span_b=52;          // period of Senkou Span B
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
```

```
double         Tenkan_sen_Buffer[];
double         Kijun_sen_Buffer[];
double         Senkou_Span_A_Buffer[];
double         Senkou_Span_B_Buffer[];
double         Chinkou_Span_Buffer[];
//--- variable for storing the handle of the iIchimoku indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Ichimoku Kinko Hyo indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,Tenkan_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(1,Kijun_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(2,Senkou_Span_A_Buffer,INDICATOR_DATA);
   SetIndexBuffer(3,Senkou_Span_B_Buffer,INDICATOR_DATA);
   SetIndexBuffer(4,Chinkou_Span_Buffer,INDICATOR_DATA);
//--- set the shift for the Senkou Span channel of kijun_sen bars in the future direct
   PlotIndexSetInteger(2,PLOT_SHIFT,kijun_sen);
//--- setting a shift for the Chikou Span line is not required, since the Chinkou data
//--- is already stored with a shift in iIchimoku
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iIchimoku)
      handle=iIchimoku(name,period,tenkan_sen,kijun_sen,senkou_span_b);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[3];
      //--- periods and shifts of the Alligator lines
      pars[0].type=TYPE_INT;
      pars[0].integer_value=tenkan_sen;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=kijun_sen;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=senkou_span_b;
      //--- create handle
      handle=IndicatorCreate(name,period,IND_ICHIMOKU,3,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iIchimoku indicator for the symbol %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Ichimoku Kinko Hyo indicator is calculated for
   short_name=StringFormat("iIchimoku(%s/%s, %d, %d ,%d)",name,EnumToString(period),
                           tenkan_sen,kijun_sen,senkou_span_b);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iIchimoku indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
```

```
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the Tenkan_sen_Buffer array is greater than the number of values in the
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the Ichimoku Kinko Hyo indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(Tenkan_sen_Buffer,Kijun_sen_Buffer,Senkou_Span_A_Buffer,
      kijun_sen,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Ichimoku Kinko Hyo indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iIchimoku indicator           |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &tenkan_sen_buffer[],     // indicator buffer of the
                           double &kijun_sen_buffer[],      // indicator buffer of the
                           double &senkou_span_A_buffer[],  // indicator buffer of the
                           double &senkou_span_B_buffer[],  // indicator buffer of the
                           double &chinkou_span_buffer[],   // indicator buffer of the
                           int senkou_span_shift,           // shift of the Senkou Spa
                           int ind_handle,                  // handle of the iIchimoku
                           int amount                       // number of copied value
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the Tenkan_sen_Buffer array with values from the indicator buffe
   if(CopyBuffer(ind_handle,0,0,amount,tenkan_sen_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("1.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the Kijun_sen_Buffer array with values from the indicator buffer
   if(CopyBuffer(ind_handle,1,0,amount,kijun_sen_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("2.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the Chinkou_Span_Buffer array with values from the indicator buf
//--- if senkou_span_shift>0, the line is shifted in the future direction by senkou_sp
   if(CopyBuffer(ind_handle,2,-senkou_span_shift,amount,senkou_span_A_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("3.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the Senkou_Span_A_Buffer array with values from the indicator bu
//--- if senkou_span_shift>0, the line is shifted in the future direction by senkou_sp
   if(CopyBuffer(ind_handle,3,-senkou_span_shift,amount,senkou_span_B_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("4.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }

//--- fill a part of the Senkou_Span_B_Buffer array with values from the indicator bu
//--- when copying Chinkou Span, we don't need to consider the shift, since the Chinko
//--- is already stored with a shift in iIchimoku
   if(CopyBuffer(ind_handle,4,0,amount,chinkou_span_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("5.Failed to copy data from the iIchimoku indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iBWMFI

The function returns the handle of the Market Facilitation Index indicator. It has only one buffer.

```
int  iBWMFI(
   string               symbol,            // symbol name
   ENUM_TIMEFRAMES      period,            // period
   ENUM_APPLIED_VOLUME  applied_volume     // volume type for calculation
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be one of the constants of ENUM_APPLIED_VOLUME.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iBWMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iBWMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- the iBWMFI plot
#property indicator_label1  "iBWMFI"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrLime,clrSaddleBrown,clrBlue,clrPink
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBWMFI,            // use iBWMFI
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iBWMFI;           // type of function
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;// type of volume
input string               symbol=" ";                 // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;      // timeframe
//--- indicator buffer
double         iBWMFIBuffer[];
double         iBWMFIColors[];
//--- variable for storing the handle of the iBWMFI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Market Facilitation Index by Bill Will
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,iBWMFIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iBWMFIColors,INDICATOR_COLOR_INDEX);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iBWMFI)
      handle=iBWMFI(name,period,applied_volume);
   else
```

```
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- type of volume
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_BWMFI,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iBWMFI indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Market Facilitation Index by Bill Williams indicat
   short_name=StringFormat("iBWMFI(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iBWMFI indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
```

```
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculatec
     {
      //--- if the iBWMFIBuffer array is greater than the number of values in the iBWN
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the indicator Market Facilitation Index by Bill W
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(iBWMFIBuffer,iBWMFIColors,handle,values_to_copy)) return
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Market Facilitation Index by Bill Williams
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iBWMFI indicator              |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &values[],    // indicator buffer for the histogram
                           double &colors[],    // indicator buffer of the histogram c
                           int ind_handle,      // handle of the iBWMFI indicator
                           int amount           // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iBWMFIBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
```

```
//--- fill a part of the iBWMFIColors array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,0,amount,colors)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iBWMFI indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iMomentum

The function returns the handle of the Momentum indicator. It has only one buffer.

```
int  iMomentum(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 mom_period,        // averaging period
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

mom_period

[in] Averaging period (bars count) for the calculation of the price change.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                               Demo_iMomentum.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMomentum technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Momentum.'

#property indicator_separate_window
```

```
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iMomentum
#property indicator_label1  "iMomentum"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMomentum,         // use iMomentum
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iMomentum;       // type of the function
input int                  mom_period=14;             // period of Momentum
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iMomentumBuffer[];
//--- variable for storing the handle of the iMomentum indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Momentum indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iMomentumBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iMomentum)
      handle=iMomentum(name,period,mom_period,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=mom_period;
      //--- type of price
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MOMENTUM,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iMomentum indicator for the symbol
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Momentum indicator is calculated for
   short_name=StringFormat("iMomentum(%s/%s, %d, %s)",name,EnumToString(period),
                           mom_period, EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iMomentum indicator
```

```
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iMomentumBuffer array is greater than the number of values in the i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iMomentumBuffer array with values of the Momentum indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iMomentumBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Momentum indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iMomentum indicator           |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Momentum values
                         int ind_handle,    // handle of the iMomentum indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iMomentumBuffer array with values from the indicator buffer t
```

```
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iMomentum indicator, error code %d",Ge
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iMFI

The function returns the handle of the Money Flow Index indicator.

```
int  iMFI(
   string               symbol,             // symbol name
   ENUM_TIMEFRAMES      period,             // period
   int                  ma_period,          // averaging period
   ENUM_APPLIED_VOLUME  applied_volume      // volume type for calculation
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for the calculation.

applied_volume

[in] The volume used. Can be any of the ENUM_APPLIED_VOLUME values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMFI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Money Flo

#property indicator_separate_window
#property indicator_buffers 1
```

```
#property indicator_plots   1
//--- the iMFI plot
#property indicator_label1  "iMFI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- horizontal levels in the indicator window
#property indicator_level1  20
#property indicator_level2  80
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMFI,              // use iMFI
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iMFI;               // type of the function
input int                  ma_period=14;                 // period
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // type of volume
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffer
double         iMFIBuffer[];
//--- variable for storing the handle of the iMFI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Money Flow Index indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iMFIBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iMFI)
      handle=iMFI(name,period,ma_period,applied_volume);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- type of volume
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_MFI,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iMFI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Money Flow Index indicator is calculated for
   short_name=StringFormat("iMFI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period, EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iMFI indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iMFIBuffer array is greater than the number of values in the iMFI
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iMFIBuffer array with values of the Money Flow Index indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iMFIBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Money Flow Index indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iMFI indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // indicator buffer of Money Flow Index va
                         int ind_handle,    // handle of the iMFI indicator
                         int amount         // number of copied values
                         )
  {
//--- reset error code
```

```
   ResetLastError();
//--- fill a part of the iMFIBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iMFI indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iMA

The function returns the handle of the Moving Average indicator. It has only one buffer.

```
int  iMA(
   string               symbol,            // symbol name
   ENUM_TIMEFRAMES      period,            // period
   int                  ma_period,         // averaging period
   int                  ma_shift,          // horizontal shift
   ENUM_MA_METHOD       ma_method,         // smoothing type
   ENUM_APPLIED_PRICE   applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the calculation of the moving average.

ma_shift

[in] Shift of the indicator relative to the price chart.

ma_method

[in] Smoothing type. Can be one of the ENUM_MA_METHOD values.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Moving Average."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iMA plot
#property indicator_label1  "iMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMA,               // use iMA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iMA;               // type of the function
input int                  ma_period=10;                // period of ma
input int                  ma_shift=0;                  // shift
input ENUM_MA_METHOD       ma_method=MODE_SMA;          // type of smoothing
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE;   // type of price
input string               symbol=" ";                  // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- indicator buffer
double         iMABuffer[];
//--- variable for storing the handle of the iMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Average indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iMABuffer,INDICATOR_DATA);
```

```
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iMA)
      handle=iMA(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of smoothing
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MA,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iMA indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Moving Average indicator is calculated for
   short_name=StringFormat("iMA(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                           ma_period, ma_shift,EnumToString(ma_method),EnumToString(a
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
```

```
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iMABuffer array is greater than the number of values in the iMA in
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the iMABuffer array with values of the Moving Average indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
```

```
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Moving Average indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the MA indicator                  |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // indicator buffer of Moving Average val
                         int shift,          // shift
                         int ind_handle,     // handle of the iMA indicator
                         int amount          // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iMABuffer array with values from the indicator buffer that ha
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iMA indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iOsMA

The function returns the handle of the Moving Average of Oscillator indicator. The OsMA oscillator shows the difference between values of MACD and its signal line. It has only one buffer.

```
int  iOsMA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 fast_ema_period,   // period for Fast Moving Average
   int                 slow_ema_period,   // period for Slow Moving Average
   int                 signal_period,     // averaging period for their difference
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

fast_ema_period

[in] Period for Fast Moving Average calculation.

slow_ema_period

[in] Period for Slow Moving Average calculation.

signal_period

[in] Averaging period for signal line calculation.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

In some systems this oscillator is also known as MACD histogram.

Example:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iOsMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOsMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Moving Ave

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iOsMA plot
#property indicator_label1  "iOsMA"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOsMA,             // use iOsMA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iOsMA;           // type of the function
input int                  fast_ema_period=12;        // period of fast ma
input int                  slow_ema_period=26;        // period of slow ma
input int                  signal_period=9;           // period of averaging of differ
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iOsMABuffer[];
//--- variable for storing the handle of the iAMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Average indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iOsMABuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iOsMA)
      handle=iOsMA(name,period,fast_ema_period,slow_ema_period,signal_period,applied_
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period of fast ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- period of slow ma
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- period of averaging of difference between the fast and the slow moving ave
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_OSMA,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create a handle of iOsMA for the pair %s/%s, error code 
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Moving Average of Oscillator indicator is calculat
   short_name=StringFormat("iOsMA(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                            fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iOsMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iOsMABuffer array is greater than the number of values in the iOsMA
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iOsMA indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iOsMABuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Moving Average of Oscillator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iOsMA indicator               |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  // indicator buffer of OsMA values
                         int ind_handle,        // handle of the iOsMA indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iOsMABuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,ama_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iOsMA indicator, error code %d",GetLas
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iMACD

The function returns the handle of the Moving Averages Convergence/Divergence indicator. In systems where OsMA is called MACD Histogram, this indicator is shown as two lines. In the client terminal the Moving Averages Convergence/Divergence looks like a histogram.

```
int  iMACD(
   string              symbol,              // symbol name
   ENUM_TIMEFRAMES     period,              // period
   int                 fast_ema_period,     // period for Fast average calculation
   int                 slow_ema_period,     // period for Slow average calculation
   int                 signal_period,       // period for their difference averaging
   ENUM_APPLIED_PRICE  applied_price        // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

fast_ema_period

[in] Period for Fast Moving Average calculation.

slow_ema_period

[in] Period for Slow Moving Average calculation.

signal_period

[in] Period for Signal line calculation.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iMACD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iMACD technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard MACD."

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- the MACD plot
#property indicator_label1  "MACD"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Signal plot
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMACD,             // use iMACD
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iMACD;           // type of the function
input int                  fast_ema_period=12;        // period of fast ma
input int                  slow_ema_period=26;        // period of slow ma
input int                  signal_period=9;           // period of averaging of differ
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffers
double         MACDBuffer[];
double         SignalBuffer[];
//--- variable for storing the handle of the iMACD indicator
int    handle;
//--- variable for storing
```

```
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Moving Averages Convergence/Divergence
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,MACDBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iMACD)
      handle=iMACD(name,period,fast_ema_period,slow_ema_period,signal_period,applied_
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period of fast ma
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- period of slow ma
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- period of averaging of difference between the fast and the slow moving ave
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MACD,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iMACD indicator for the symbol %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Moving Average Convergence/Divergence indicator is
   short_name=StringFormat("iMACD(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                           fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iMACD indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the MACDBuffer array is greater than the number of values in the iMACD
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iMACD indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(MACDBuffer,SignalBuffer,handle,values_to_copy)) return(0)
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Moving Averages indicator Convergence/Diver
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iMACD indicator               |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &macd_buffer[],    // indicator buffer of MACD value
                           double &signal_buffer[],  // indicator buffer of the signal
                           int ind_handle,           // handle of the iMACD indicator
                           int amount                // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iMACDBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,macd_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLas
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }

//--- fill a part of the SignalBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iMACD indicator, error code %d",GetLas
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iOBV

The function returns the handle of the On Balance Volume indicator. It has only one buffer.

```
int  iOBV(
   string               symbol,             // symbol name
   ENUM_TIMEFRAMES      period,             // period
   ENUM_APPLIED_VOLUME  applied_volume      // volume type for calculation
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be any of the ENUM_APPLIED_VOLUME values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iOBV.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iOBV technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---  iOBV
#property indicator_label1  "iOBV"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOBV ,             // use iOBV
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iOBV;               // type of the function
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // type of volume
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffers
double         iOBVBuffer[];
//--- variable for storing the handle of the iOBV indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the On Balance Volume indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iOBVBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iOBV)
      handle=iOBV(name,period,applied_volume);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- type of volume
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_OBV,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iOBV indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the On Balance Volume indicator is calculated for
   short_name=StringFormat("iOBV(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iOBV indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iOBVBuffer array is greater than the number of values in the iOBV
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iOBV indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iOBVBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the On Balance Volume indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iOBV indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &obv_buffer[],  // indicator buffer of OBV values
                         int ind_handle,        // handle of the iOBV indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iOBVBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,obv_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iOBV indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iSAR

The function returns the handle of the Parabolic Stop and Reverse system indicator. It has only one buffer.

```
int  iSAR(
   string           symbol,      // symbol name
   ENUM_TIMEFRAMES  period,      // period
   double           step,        // price increment step - acceleration factor
   double           maximum      // maximum value of step
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

step

[in] The step of price increment, usually 0.02.

maximum

[in] The maximum step, usually 0.2.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iSAR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iSAR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Parabolic

#property indicator_chart_window
```

```
#property indicator_buffers 1
#property indicator_plots   1
//--- drawing iSAR
#property indicator_label1  "iSAR"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iSAR,              // use iSAR
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iSAR;               // type of the function
input double               step=0.02;                    // step - the acceleration fa
input double               maximum=0.2;                  // maximum value of step
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffers
double         iSARBuffer[];
//--- variable for storing the handle of the iSAR indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Parabolic SAR indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iSARBuffer,INDICATOR_DATA);
//--- set a symbol code from the Wingdings charset for the PLOT_ARROW property for dis
   PlotIndexSetInteger(0,PLOT_ARROW,159);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iSAR)
      handle=iSAR(name,period,step,maximum);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- step value
      pars[0].type=TYPE_DOUBLE;
      pars[0].double_value=step;
      //--- limit of the step value that can be used for calculations
      pars[1].type=TYPE_DOUBLE;
      pars[1].double_value=maximum;
      handle=IndicatorCreate(name,period,IND_SAR,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iSAR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Parabolic SAR indicator is calculated for
   short_name=StringFormat("iSAR(%s/%s, %G, %G)",name,EnumToString(period),
                           step,maximum);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iSAR indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iSARBuffer array is greater than the number of values in the iSAR
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iSAR indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iSARBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Parabolic SAR indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iSAR indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &sar_buffer[],  // indicator buffer of Parabolic SAR
                         int ind_handle,        // handle of the iSAR indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
```

```
   ResetLastError();
//--- fill a part of the iSARBuffer array with values from the indicator buffer that 
   if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iSAR indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iRSI

The function returns the handle of the Relative Strength Index indicator. It has only one buffer.

```
int  iRSI(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the RSI calculation.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iRSI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRSI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Relative 

#property indicator_separate_window
```

```
#property indicator_buffers 1
#property indicator_plots   1
//--- drawing iRSI
#property indicator_label1  "iRSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- limits for displaying of values in the indicator window
#property indicator_maximum 100
#property indicator_minimum 0
//--- horizontal levels in the indicator window
#property indicator_level1  70.0
#property indicator_level2  30.0
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRSI,              // use iRSI
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iRSI;               // type of the function
input int                  ma_period=14;                 // period of averaging
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE;    // type of price
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffer
double         iRSIBuffer[];
//--- variable for storing the handle of the iRSI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Relative Strength Index indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iRSIBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iRSI)
      handle=iRSI(name,period,ma_period,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- period of moving average
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- limit of the step value that can be used for calculations
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_RSI,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iRSI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Relative Strength Index indicator is calculated fo
   short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),
                           ma_period,applied_price);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//--- number of values copied from the iRSI indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means 
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iRSIBuffer array is greater than the number of values in the iRSI i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the iRSI indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iRSIBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Relative Strength Index indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iRSI indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &rsi_buffer[],  // indicator buffer of Relative Streng
                         int ind_handle,        // handle of the iRSI indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iRSIBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,0,amount,rsi_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iRSI indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iRVI

The function returns the handle of the Relative Vigor Index indicator.

```
int  iRVI(
   string           symbol,        // symbol name
   ENUM_TIMEFRAMES  period,        // period
   int              ma_period      // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the RVI calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers are the following: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iRVI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iRVI technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Relative

#property indicator_separate_window
#property indicator_buffers 2
```

```
#property indicator_plots   2
//--- the RVI plot
#property indicator_label1  "RVI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Signal plot
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRVI,              // use iRVI
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iRVI;               // type of the function
input int                  ma_period=10;                 // period for calculations
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffers
double         RVIBuffer[];
double         SignalBuffer[];
//--- variable for storing the handle of the iRVI indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Relative Vigor Index indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,RVIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iRVI)
      handle=iRVI(name,period,ma_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- period for calculations
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_RVI,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iRVI indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Relative Vigor Index indicator is calculated for
   short_name=StringFormat("iRVI(%s/%s, %d, %d)",name,EnumToString(period),ma_period);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iRVI indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means 
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the RVIBuffer array is greater than the number of values in the iRVI in
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iRVI indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(RVIBuffer,SignalBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Relative Vigor Index indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iRVI indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &rvi_buffer[],     // indicator buffer of Relative Vig
                         double &signal_buffer[],  // indicator buffer of the signal I
                         int ind_handle,           // handle of the iRVI indicator
                         int amount                // number of copied values
                         )
  {
```

```
//--- reset error code
   ResetLastError();
//--- fill a part of the iRVIBuffer array with values from the indicator buffer that h
   if(CopyBuffer(ind_handle,0,0,amount,rvi_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- fill a part of the SignalBuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iRVI indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iStdDev

The function returns the handle of the Standard Deviation indicator. It has only one buffer.

```
int  iStdDev(
   string               symbol,            // symbol name
   ENUM_TIMEFRAMES      period,            // period
   int                  ma_period,         // averaging period
   int                  ma_shift,          // horizontal shift
   ENUM_MA_METHOD       ma_method,         // smoothing type
   ENUM_APPLIED_PRICE   applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period for the indicator calculations.

ma_shift

[in] Shift of the indicator relative to the price chart.

ma_method

[in] Type of averaging. Can be any of the ENUM_MA_METHOD values.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iStdDev.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStdDev technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the normal Standard Dev

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iStdDev plot
#property indicator_label1  "iStdDev"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStdDev,            // use iStdDev
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iStdDev;         // type of the function
input int                  ma_period=20;              // period of averaging
input int                  ma_shift=0;                // shift
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // type of smoothing
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iStdDevBuffer[];
//--- variable for storing the handle of the iStdDev indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Standard Deviation indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iStdDevBuffer,INDICATOR_DATA);
```

```
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iStdDev)
      handle=iStdDev(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of smoothing
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_STDDEV,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iStdDev indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Standard Deviation indicator is calculated for
   short_name=StringFormat("iStdDev(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(ma_method),EnumToString(ap
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
```

```
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iStdDev indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iStdDevBuffer array is greater than the number of values in the iS
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the Standard Deviation indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iStdDevBuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
```

```
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Standard Deviation indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iStdDev indicator             |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &std_buffer[],  // indicator buffer of the Standard De
                         int std_shift,         // shift of the Standard Deviation lin
                         int ind_handle,        // handle of the iStdDev indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iStdDevBuffer array with values from the indicator buffer tha
   if(CopyBuffer(ind_handle,0,-std_shift,amount,std_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iStdDev indicator, error code %d",GetL
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iStochastic

The function returns the handle of the Stochastic Oscillator indicator.

```
int  iStochastic(
   string            symbol,          // symbol name
   ENUM_TIMEFRAMES   period,          // period
   int               Kperiod,         // K-period (number of bars for calculations)
   int               Dperiod,         // D-period (period of first smoothing)
   int               slowing,         // final smoothing
   ENUM_MA_METHOD    ma_method,       // type of smoothing
   ENUM_STO_PRICE    price_field      // stochastic calculation method
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

Kperiod

[in] Averaging period (bars count) for the %K line calculation.

Dperiod

[in] Averaging period (bars count) for the %D line calculation.

slowing

[in] Slowing value.

ma_method

[in] Type of averaging. Can be any of the ENUM_MA_METHOD values.

price_field

[in] Parameter of price selection for calculations. Can be one of the ENUM_STO_PRICE values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Note

The buffer numbers: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_iStochastic.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iStochastic technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Stochasti

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- the Stochastic plot
#property indicator_label1  "Stochastic"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- the Signal plot
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- set limit of the indicator values
#property indicator_minimum 0
#property indicator_maximum 100
//--- horizontal levels in the indicator window
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStochastic,       // use iStochastic
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iStochastic;     // type of the function
input int                  Kperiod=5;                 // the K period (the number of
input int                  Dperiod=3;                 // the D period (the period of
input int                  slowing=3;                 // period of final smoothing
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // type of smoothing
input ENUM_STO_PRICE       price_field=STO_LOWHIGH;   // method of calculation of the
input string                symbol=" ";                 // symbol
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;      // timeframe
//--- indicator buffers
double          StochasticBuffer[];
double          SignalBuffer[];
//--- variable for storing the handle of the iStochastic indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Stochastic Oscillator indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of arrays to indicator buffers
   SetIndexBuffer(0,StochasticBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iStochastic)
      handle=iStochastic(name,period,Kperiod,Dperiod,slowing,ma_method,price_field);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[5];
      //--- the K period for calculations
      pars[0].type=TYPE_INT;
      pars[0].integer_value=Kperiod;
      //--- the D period for primary smoothing
      pars[1].type=TYPE_INT;
      pars[1].integer_value=Dperiod;
      //--- the K period for final smoothing
      pars[2].type=TYPE_INT;
      pars[2].integer_value=slowing;
      //--- type of smoothing
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ma_method;
      //--- method of calculation of the Stochastic
      pars[4].type=TYPE_INT;
      pars[4].integer_value=price_field;
      handle=IndicatorCreate(name,period,IND_STOCHASTIC,5,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iStochastic indicator for the symbo
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Stochastic Oscillator indicator is calculated for
   short_name=StringFormat("iStochastic(%s/%s, %d, %d, %d, %s, %s)",name,EnumToString
                           Kperiod,Dperiod,slowing,EnumToString(ma_method),EnumToStri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iStochastic indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErr
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
```

```
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the StochasticBuffer array is greater than the number of values in the
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iStochastic indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(StochasticBuffer,SignalBuffer,handle,values_to_copy)) ret
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Stochastic Oscillator indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iStochastic indicator         |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &main_buffer[],    // indicator buffer of Stochasti
                           double &signal_buffer[],  // indicator buffer of the signal
                           int ind_handle,           // handle of the iStochastic ind
                           int amount                // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the StochasticBuffer array with values from the indicator buffer
   if(CopyBuffer(ind_handle,MAIN_LINE,0,amount,main_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- fill a part of the SignalBuffer array with values from the indicator buffer that
```

```
   if(CopyBuffer(ind_handle,SIGNAL_LINE,0,amount,signal_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iStochastic indicator, error code %d",
      //--- quit with zero result - it means that the indicator is considered as not 
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iTEMA

The function returns the handle of the Triple Exponential Moving Average indicator. It has only one buffer.

```
int  iTEMA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   int                 ma_shift,          // horizontal shift of indicator
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculation.

ma_shift

[in] Shift of indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iTEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTEMA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
```

```
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All the other parameters are similar to the standard Triple Exp

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iTEMA plot
#property indicator_label1  "iTEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTEMA,              // use iTEMA
   Call_IndicatorCreate     // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iTEMA;           // type of the function
input int                  ma_period=14;              // period of averaging
input int                  ma_shift=0;                // shift
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iTEMABuffer[];
//--- variable for storing the handle of the iTEMA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Triple Exponential Moving Average indic
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iTEMABuffer,INDICATOR_DATA);
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
```

```
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iTEMA)
      handle=iTEMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[3];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- shift
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- type of price
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_TEMA,3,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iTEMA indicator for the symbol %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Triple Exponential Moving Average indicator is cal
   short_name=StringFormat("iTEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                 const int prev_calculated,
                 const datetime &time[],
                 const double &open[],
                 const double &high[],
                 const double &low[],
                 const double &close[],
                 const long &tick_volume[],
                 const long &volume[],
                 const int &spread[])
  {
//--- number of values copied from the iTEMA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iTEMABuffer array is greater than the number of values in the iTEMA
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values from the Triple Exponential Moving Average indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iTEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Triple Exponential Moving Average indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iTEMA indicator               |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &tema_buffer[], // indicator buffer of Triple Exponent
                         int t_shift,           // shift of the line
                         int ind_handle,        // handle of the iTEMA indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iTEMABuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-t_shift,amount,tema_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iTEMA indicator, error code %d",GetLas
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iTriX

The function returns the handle of the Triple Exponential Moving Averages Oscillator indicator. It has only one buffer.

```
int  iTriX(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 ma_period,         // averaging period
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

ma_period

[in] Averaging period (bars count) for calculations.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iTriX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iTriX technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
```

```
#property indicator_buffers 1
#property indicator_plots   1
//--- the iTriX plot
#property indicator_label1  "iTriX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTriX,             // use iTriX
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iTriX;           // type of the function
input int                  ma_period=14;              // period
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iTriXBuffer[];
//--- variable for storing the handle of the iTriX indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Triple Exponential Moving Averages Osc
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iTriXBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iTriX)
      handle=iTriX(name,period,ma_period,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[2];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- type of price
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_TRIX,2,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iTriX indicator for the symbol %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Triple Exponential Moving Averages Oscillator is c
   short_name=StringFormat("iTriX(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iTriX indicator
```

```
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- if the iTriXBuffer array is greater than the number of values in the iTri
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values from the Triple Exponential Moving Averages Oscillat
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iTriXBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Triple Exponential Moving Averages Oscillat
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iTriX indicator               |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &trix_buffer[], // indicator buffer of values of Tripl
                         int ind_handle,        // handle of the iTriX indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iTriXBuffer array with values from the indicator buffer that
```

```
    if(CopyBuffer(ind_handle,0,0,amount,trix_buffer)<0)
      {
       //--- if the copying fails, tell the error code
       PrintFormat("Failed to copy data from the iTriX indicator, error code %d",GetLas
       //--- quit with zero result - it means that the indicator is considered as not c
       return(false);
      }
//--- everything is fine
    return(true);
   }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
   {
    if(handle!=INVALID_HANDLE)
       IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
    Comment("");
   }
```

iWPR

The function returns the handle of the Larry Williams' Percent Range indicator. It has only one buffer.

```
int  iWPR(
   string           symbol,          // symbol name
   ENUM_TIMEFRAMES  period,          // period
   int              calc_period      // averaging period
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

calc_period

[in] Period (bars count) for the indicator calculation.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iWPR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iWPR technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iWPR plot
#property indicator_label1  "iWPR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrCyan
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- set limit of the indicator values
#property indicator_minimum -100
#property indicator_maximum 0
//--- horizontal levels in the indicator window
#property indicator_level1  -20.0
#property indicator_level2  -80.0
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iWPR,              // use iWPR
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iWPR;            // type of the function
input int                  calc_period=14;            // period
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iWPRBuffer[];
//--- variable for storing the handle of the iWPR indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Larry Williams' Percent Range indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iWPRBuffer,INDICATOR_DATA);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
```

```
//--- create handle of the indicator
   if(type==Call_iWPR)
      handle=iWPR(name,period,calc_period);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[1];
      //--- period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=calc_period;
      handle=IndicatorCreate(name,period,IND_WPR,1,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iWPR indicator for the symbol %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Williams' Percent Range indicator is calculated fo
   short_name=StringFormat("iWPR(%s/%s, %d)",name,EnumToString(period),calc_period);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iWPR indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iWPRBuffer array is greater than the number of values in the iWPR i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values of the Williams' Percent Range indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iWPRBuffer,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Larry Williams' Percent Range indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iWPR indicator                |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &wpr_buffer[],  // indicator buffer of Williams' Perce
                         int ind_handle,        // handle of the iWPR indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iWPRBuffer array with values from the indicator buffer that h
   if(CopyBuffer(ind_handle,0,0,amount,wpr_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iWPR indicator, error code %d",GetLast
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iVIDyA

The function returns the handle of the Variable Index Dynamic Average indicator. It has only one buffer.

```
int  iVIDyA(
   string              symbol,            // symbol name
   ENUM_TIMEFRAMES     period,            // period
   int                 cmo_period,        // period for Chande Momentum
   int                 ema_period,        // EMA smoothing period
   int                 ma_shift,          // horizontal shift on the price chart
   ENUM_APPLIED_PRICE  applied_price      // type of price or handle
   );
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

cmo_period

[in] Period (bars count) for the Chande Momentum Oscillator calculation.

ema_period

[in] EMA period (bars count) for smoothing factor calculation.

ma_shift

[in] Shift of the indicator relative to the price chart.

applied_price

[in] The price used. Can be any of the price constants ENUM_APPLIED_PRICE or a handle of another indicator.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iVIDyA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVIDyA technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'
#property description "All other parameters like in the standard Variable Index Dynam

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- the iVIDyA plot
#property indicator_label1  "iVIDyA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                     |
//+-------------------------------------------------------------------+
enum Creation
  {
   Call_iVIDyA,            // use iVIDyA
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iVIDyA;          // type of the function
input int                  cmo_period=15;             // the Chande Momentum period
input int                  ema_period=12;             // period of the factor of smoot
input int                  ma_shift=0;                // shift
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // type of price
input string               symbol=" ";                // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // timeframe
//--- indicator buffer
double         iVIDyABuffer[];
//--- variable for storing the handle of the iVIDyA indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Variable Index Dynamic Average indicat
int    bars_calculated=0;
//+-------------------------------------------------------------------+
//| Custom indicator initialization function                          |
//+-------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iVIDyABuffer,INDICATOR_DATA);
```

```
//--- set shift
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iVIDyA)
      handle=iVIDyA(name,period,cmo_period,ema_period,ma_shift,applied_price);
   else
     {
      //--- fill the structure with parameters of the indicator
      MqlParam pars[4];
      //--- the Chande Momentum period
      pars[0].type=TYPE_INT;
      pars[0].integer_value=cmo_period;
      //--- period of the factor of smoothing
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ema_period;
      //--- shift
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_shift;
      //--- type of price
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_VIDYA,4,pars);
     }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iVIDyA indicator for the symbol %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Variable Index Dynamic Average indicator is calcul
   short_name=StringFormat("iVIDyA(%s/%s, %d, %d, %d, %s)",name,EnumToString(period),
                           cmo_period,ema_period,ma_shift,EnumToString(applied_price)
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
```

```
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iVIDyA indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of val
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iVIDyABuffer array is greater than the number of values in the iVII
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and si
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the array with values from the Variable Index Dynamic Average indicator
//--- if FillArrayFromBuffer returns false, it means the information is nor ready yet,
   if(!FillArrayFromBuffer(iVIDyABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
```

```
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Variable Index Dynamic Average indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iVIDyA indicator              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &vidya_buffer[],// indicator buffer of Variable Index
                         int v_shift,           // shift of the line
                         int ind_handle,        // handle of the iVIDyA indicator
                         int amount             // number of copied values
                         )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iVIDyABuffer array with values from the indicator buffer that
   if(CopyBuffer(ind_handle,0,-v_shift,amount,vidya_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iVIDyA indicator, error code %d",GetLa
      //--- quit with zero result - it means that the indicator is considered as not
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

iVolumes

The function returns the handle of the Volumes indicator. It has an only one buffer.

```
int  iVolumes(
   string              symbol,             // symbol name
   ENUM_TIMEFRAMES     period,             // period
   ENUM_APPLIED_VOLUME applied_volume      // volume type for calculation
   )
```

Parameters

symbol

[in] The symbol name of the security, the data of which should be used to calculate the indicator. The NULL value means the current symbol.

period

[in] The value of the period can be one of the ENUM_TIMEFRAMES values, 0 means the current timeframe.

applied_volume

[in] The volume used. Can be any of the ENUM_APPLIED_VOLUME values.

Return Value

Returns the handle of a specified technical indicator, in case of failure returns INVALID_HANDLE. The computer memory can be freed from an indicator that is no more utilized, using the IndicatorRelease() function, to which the indicator handle is passed.

Example:

```
//+------------------------------------------------------------------+
//|                                              Demo_iVolumes.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "The indicator demonstrates how to obtain data"
#property description "of indicator buffers for the iVolumes technical indicator."
#property description "A symbol and timeframe used for calculation of the indicator,"
#property description "are set by the symbol and period parameters."
#property description "The method of creation of the handle is set through the 'type'

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- the iVolumes plot
#property indicator_label1  "iVolumes"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//| Enumeration of the methods of handle creation                    |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iVolumes,          // use iVolumes
   Call_IndicatorCreate    // use IndicatorCreate
  };
//--- input parameters
input Creation             type=Call_iVolumes;           // type of the function
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // type of volume
input string               symbol=" ";                   // symbol
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- indicator buffers
double         iVolumesBuffer[];
double         iVolumesColors[];
//--- variable for storing the handle of the iVolumes indicator
int    handle;
//--- variable for storing
string name=symbol;
//--- name of the indicator on a chart
string short_name;
//--- we will keep the number of values in the Volumes indicator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- assignment of array to indicator buffer
   SetIndexBuffer(0,iVolumesBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iVolumesColors,INDICATOR_COLOR_INDEX);
//--- determine the symbol the indicator is drawn for
   name=symbol;
//--- delete spaces to the right and to the left
   StringTrimRight(name);
   StringTrimLeft(name);
//--- if it results in zero length of the 'name' string
   if(StringLen(name)==0)
     {
      //--- take the symbol of the chart the indicator is attached to
      name=_Symbol;
     }
//--- create handle of the indicator
   if(type==Call_iVolumes)
      handle=iVolumes(name,period,applied_volume);
   else
```

```
      {
       //--- fill the structure with parameters of the indicator
       MqlParam pars[1];
       //--- type of price
       pars[0].type=TYPE_INT;
       pars[0].integer_value=applied_volume;
       handle=IndicatorCreate(name,period,IND_VOLUMES,1,pars);
      }
//--- if the handle is not created
   if(handle==INVALID_HANDLE)
     {
      //--- tell about the failure and output the error code
      PrintFormat("Failed to create handle of the iVolumes indicator for the symbol %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- the indicator is stopped early
      return(INIT_FAILED);
     }
//--- show the symbol/timeframe the Volumes indicator is calculated for
   short_name=StringFormat("iVolumes(%s/%s, %s)",name,EnumToString(period),EnumToStrin
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- normal initialization of the indicator
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- number of values copied from the iVolumes indicator
   int values_to_copy;
//--- determine the number of values calculated in the indicator
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() returned %d, error code %d",calculated,GetLastErro
      return(0);
     }
//--- if it is the first start of calculation of the indicator or if the number of va
```

```
//---or if it is necessary to calculated the indicator for two or more bars (it means
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
     {
      //--- if the iVolumesBuffer array is greater than the number of values in the i
      //--- otherwise, we copy less than the size of indicator buffers
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- it means that it's not the first time of the indicator calculation, and s
      //--- for calculation not more than one bar is added
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- fill the arrays with values of the iVolumes indicator
//--- if FillArraysFromBuffer returns false, it means the information is nor ready yet
   if(!FillArraysFromBuffers(iVolumesBuffer,iVolumesColors,handle,values_to_copy)) ret
//--- form the message
   string comm=StringFormat("%s ==>  Updated value in the indicator %s: %d",
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- display the service message on the chart
   Comment(comm);
//--- memorize the number of values in the Volumes indicator
   bars_calculated=calculated;
//--- return the prev_calculated value for the next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Filling indicator buffers from the iVolumes indicator            |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &volume_buffer[],    // indicator buffer of Volumes v
                           double &color_buffer[],     // indicator buffer of colors
                           int ind_handle,             // handle of the iVolumes indica
                           int amount                  // number of copied values
                           )
  {
//--- reset error code
   ResetLastError();
//--- fill a part of the iVolumesBuffer array with values from the indicator buffer th
   if(CopyBuffer(ind_handle,0,0,amount,volume_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",Get
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- fill a part of the iVolumesColors array with values from the indicator buffer th
```

```
   if(CopyBuffer(ind_handle,1,0,amount,color_buffer)<0)
     {
      //--- if the copying fails, tell the error code
      PrintFormat("Failed to copy data from the iVolumes indicator, error code %d",Get
      //--- quit with zero result - it means that the indicator is considered as not c
      return(false);
     }
//--- everything is fine
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   if(handle!=INVALID_HANDLE)
      IndicatorRelease(handle);
//--- clear the chart after deleting the indicator
   Comment("");
  }
```

Working with Optimization Results

Functions for organizing custom processing of the optimization results in the strategy tester. They can be called during optimization in testing agents, as well as locally in Expert Advisors and scripts.

When you run an Expert Advisor in the strategy tester, you can create your own data array based on the simple types or simple structures (they do not contain strings, class objects or objects of dynamic arrays). This data set can be saved using the FrameAdd() function in a special structure called a frame. During the optimization of an Expert Advisor, each agent can send a series of frames to the terminal. All the received frames are written in the *.MQD file named as the Expert Advisor in the terminal_directory\MQL5\Files\Tester folder. They are written in the order they are received from the agents. Receipt of a frame in the client terminal from a testing agent generates the TesterPass event.

Frames can be stored in the computer memory and in a file with the specified name. The MQL5 language sets no limitations on the number of frames.

Memory and disk space limits in MQL5 Cloud Network

The following limitation applies to optimizations run in the MQL5 Cloud Network: the Expert Advisor must not write to disk more than 4GB of information or use more than 4GB of RAM. If the limit is exceeded, the network agent will not be able to complete the calculation correctly, and you will not receive the result. However, you will be charged for all the time spent on the calculations.

If you need to get information from each optimization pass, send frames without writing to disk. To avoid using file operations in Expert Advisors during calculations in the MQL5 Cloud Network, you can use the following check:

```
   int handle=INVALID_HANDLE;
   bool file_operations_allowed=true;
   if(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_FORWARD))
      file_operations_allowed=false;

   if(file_operations_allowed)
     {
      ...
      handle=FileOpen(...);
      ...
     }
```

Function	Action
FrameFirst	Moves a pointer of frame reading to the beginning and resets the previously set filter
FrameFilter	Sets the frame reading filter and moves the pointer to the beginning
FrameNext	Reads a frame and moves the pointer to the next one
FrameInputs	Receives input parameters, on which the frame is formed
FrameAdd	Adds a frame with data

Function	Action
ParameterGetRange	Receives data on the values range and the change step for an input variable when optimizing an Expert Advisor in the Strategy Tester
ParameterSetRange	Specifies the use of input variable when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values

See also

Testing Statistics, Properties of a Running MQL5 Program

FrameFirst

Moves a pointer of frame reading to the beginning and resets a set filter.

```
bool  FrameFirst();
```

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

FrameFilter

Sets the frame reading filter and moves the pointer to the beginning.

```
bool  FrameFilter(
   const string  name,          // Public name/label
   long          id             // Public ID
   );
```

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

If an empty string is passed as the first parameter, the filter will work only with a numeric parameter, i.e. only frames with the specified id will be viewed. If the value of the second parameter is ULONG_MAX, only a text filter works.

Call of FrameFilter("", ULONG_MAX) is equivalent to calling FrameFirst(), i.e. equal to not using any filter.

FrameNext

Reads a frame and moves the pointer to the next one. There are two variants of the function.

1. Calling to receive one numeric value

```
bool  FrameNext(
   ulong&    pass,       // The number of a pass in the optimization, during which the
   string&   name,       // Public name/label
   long&     id,         // Public ID
   double&   value       // Value
   );
```

2. Calling to receive all the data of a frame

```
bool  FrameNext(
   ulong&    pass,       // The number of a pass in the optimization, during which the
   string&   name,       // Public name/label
   long&     id,         // Public ID
   double&   value,      // Value
   void&     data[]      // Array of any type
   );
```

Parameters

pass

[out] The number of a pass during optimization in the strategy tester.

name

[out] The name of the identifier.

id

[out] The value of the identifier.

value

[out] A single numeric value.

data

[out] An array of any type.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

In the second version of the call, you must correctly handle the received data in the *data[]* array.

FrameInputs

Receives input parameters, on which the frame with the specified pass number is formed.

```
bool  FrameInputs(
   ulong     pass,                // The number of a pass in the optimization
   string&   parameters[],        // An array of strings of form "parameterN=valueN"
   uint&     parameters_count     // The total number of parameters
   );
```

Parameters

pass

[in] The number of a pass during optimization in the strategy tester.

parameters

[out] A string array with the description of names and parameter values

parameters_count

[out] The number of elements in the array *parameters[]*.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

Having obtained the number of strings *parameters_count* in the *parameters[]* array, you can organize a loop to go through all records. This will help you find the values of input parameters of an Expert Advisor for the specified pass number.

FrameAdd

Adds a frame with data. There are two variants of the function.

1. Adding data from a file

```
bool  FrameAdd(
   const string  name,         // Public name/label
   long          id,           // Public ID
   double        value,        // Value
   const string  filename      // Name of a data file
   );
```

2. Adding data from an array of any type

```
bool  FrameAdd(
   const string  name,         // Public name/label
   long          id,           // Public ID
   double        value,        // Value
   const void&   data[]        // Array of any type
   );
```

Parameters

name

[in] Public frame label. It can be used for a filter in the FrameFilter() function.

id

[in] A public identifier of the frame. It can be used for a filter in the FrameFilter() function.

value

[in] A numeric value to write into the frame. It is used to transmit a single pass result like in the OnTester() function.

filename

[in] The name of the file that contains data to add to the frame. The file must be locate in the folder MQL5/Files.

data

[in] An array of any type to write into the frame. Passed by reference.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

ParameterGetRange

Receives data on the values range and the change step for an input variable when optimizing an Expert Advisor in the Strategy Tester. There are 2 variants of the function.

1. Receiving data for the integer type input parameter

```
bool  ParameterGetRange(
   const string  name,          // parameter (input variable) name
   bool&         enable,        // parameter optimization enabled
   long&         value,         // parameter value
   long&         start,         // initial value
   long&         step,          // change step
   long&         stop           // final value
   );
```

2. Receiving data for the real type input parameter

```
bool  ParameterGetRange(
   const string  name,          // parameter (input variable) name
   bool&         enable,        // parameter optimization enabled
   double&       value,         // parameter value
   double&       start,         // initial value
   double&       step,          // change step
   double&       stop           // final value
   );
```

Parameters

name

[in] input variable ID. These variables are external parameters of an application. Their values can be specified when launching on a chart or during a single test.

enable

[out] Flag that this parameter can be used to enumerate the values during the optimization in the Strategy Tester.

value

[out] Parameter value.

start

[out] Initial parameter value during the optimization.

step

[out] Parameter change step when enumerating its values.

stop

[out] Final parameter value during the optimization.

Return Value

Returns true if successful, otherwise false. For information about the error, use the GetLastError() function.

Note

The function can be called only from OnTesterInit(), OnTesterPass() and OnTesterDeinit() handlers. It has been introduced to receive Expert Advisor input parameters' values and variation ranges during the optimization in the Strategy Tester.

When called in OnTesterInit(), the obtained data can be used to redefine the rules for enumeration of any input variable using ParameterSetRange() function. Therefore, new Start, Stop and Step values can be set and the input parameter can even be completely excluded from optimization regardless of the Strategy Tester settings. This allows you to manage the area of the input parameters during the optimization by excluding some parameters from the optimization depending on the Expert Advisor's key parameters' values.

Example:

```
#property description "Expert Advisor for ParameterGetRange() function demonstration."
#property description "Should be launched in the optimization mode of the Strategy Tes
//--- input parameters
input int                 Input1=1;
input double              Input2=2.0;
input bool                Input3=false;
input ENUM_DAY_OF_WEEK    Input4=SUNDAY;

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Expert Advisor is designed for operation only in the Strategy Tester
   if(!MQL5InfoInteger(MQL5_OPTIMIZATION))
     {
      MessageBox("Should be launched in the optimization mode of the Strategy Tester!"
      //--- finish the Expert Advisor operation in advance and remove from the chart
      return(INIT_FAILED);
     }
//--- successful completion of initialization
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| TesterInit function                                              |
//+------------------------------------------------------------------+
void OnTesterInit()
  {
//--- example for long type input parameter
   string name="Input1";
   bool enable;
   long par1,par1_start,par1_step,par1_stop;
   ParameterGetRange(name,enable,par1,par1_start,par1_step,par1_stop);
   Print("First parameter");
   PrintFormat("%s=%d  enable=%s  from %d to %d with step=%d",
               name,par1,(string)enable,par1_start,par1_stop,par1_step);
//--- example for double type input parameter
   name="Input2";
   double par2,par2_start,par2_step,par2_stop;
   ParameterGetRange(name,enable,par2,par2_start,par2_step,par2_stop);
   Print("Second parameter");
   PrintFormat("%s=%G  enable=%s  from %G to %G with step=%G",
               name,par2,(string)enable,par2_start,par2_stop,par2_step);

//--- example for bool type input parameter
   name="Input3";
   long par3,par3_start,par3_step,par3_stop;
   ParameterGetRange(name,enable,par3,par3_start,par3_step,par3_stop);
   Print("Third parameter");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,(string)par3,(string)enable,
               (string)par3_start,(string)par3_stop);
//--- example for enumeration type input parameter
   name="Input4";
   long par4,par4_start,par4_step,par4_stop;
   ParameterGetRange(name,enable,par4,par4_start,par4_step,par4_stop);
   Print("Fourth parameter");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,EnumToString((ENUM_DAY_OF_WEEK)par4),(string)enable,
               EnumToString((ENUM_DAY_OF_WEEK)par4_start),
               EnumToString((ENUM_DAY_OF_WEEK)par4_stop));
  }
//+------------------------------------------------------------------+
//| TesterDeinit function                                            |
//+------------------------------------------------------------------+
void OnTesterDeinit()
  {
//--- this message will be shown after optimization is complete
   Print(__FUNCTION__," Optimization completed");
  }
```

ParameterSetRange

Specifies the use of input variable when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values. There are 2 variants of the function.

1. Specifying the values for the integer type input parameter

```
bool  ParameterSetRange(
   const string  name,          // parameter (input variable) name
   bool          enable,        // parameter optimization enabled
   long          value,         // parameter value
   long          start,         // initial value
   long          step,          // change step
   long          stop           // final value
   );
```

2. Specifying the values for the real type input parameter

```
bool  ParameterSetRange(
   const string  name,          // parameter (input variable) name
   bool          enable,        // parameter optimization enabled
   double        value,         // parameter value
   double        start,         // initial value
   double        step,          // change step
   double        stop           // final value
   );
```

Parameters

name

[in] input or sinput variable ID. These variables are external parameters of an application. Their values can be specified when launching the program.

enable

[in] Enable this parameter to enumerate the values during the optimization in the Strategy Tester.

value

[in] Parameter value.

start

[in] Initial parameter value during the optimization.

step

[in] Parameter change step when enumerating its values.

stop

[in] Final parameter value during the optimization.

Return Value

Returns true if successful, otherwise false. For information about the error, use the GetLastError() function.

Note

The function can be called only from OnTesterInit() handler when launching optimization from the Strategy Tester. It is designed for specifying the parameter's range and change step. The parameter can be completely excluded from optimization regardless of the Strategy Tester settings. It also allows using the variables declared with sinput modifier in the optimization process.

ParameterSetRange() function allows you to manage an Expert Advisor optimization in the Strategy Tester depending on its key parameters' values by including or excluding required input parameters from the optimization and setting the required range and the change step.

Event Functions

This group contains functions for working with custom events and timer events. Besides this group, there are special functions for handling predefined events.

Function	Action
EventSetMillisecondTimer	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart
EventSetTimer	Starts the timer event generator with the specified periodicity for the current chart
EventKillTimer	Stops the generation of events by the timer in the current chart
EventChartCustom	Generates a custom event for the specified chart

See also

Types of Chart Events

EventSetMillisecondTimer

The function indicates to the client terminal that timer events should be generated at intervals less than one second for this Expert Advisor or indicator.

```
bool  EventSetMillisecondTimer(
   int   milliseconds       // number of milliseconds
   );
```

Parameters

milliseconds

[in] Number of milliseconds defining the frequency of timer events.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, GetLastError() function should be called.

Note

This feature is designed for the cases when high-resolution timer is required. In other words, timer events should be received more frequently than once per second. If a conventional timer with the period of more than one second is enough for you, use EventSetTimer().

In general, when the timer period is reduced, the testing time is increased, as the handler of timer events is called more often. When working in real-time mode, timer events are generated no more than 1 time in 10-16 milliseconds due to hardware limitations.

Usually, this function should be called from OnInit() function or in class constructor. To handle events coming from the timer, an Expert Advisor or an indicator should have OnTimer() function.

Each Expert Advisor and each indicator work with its own timer receiving events solely from this timer. During mql5 application shutdown, the timer is forcibly destroyed in case it has been created but has not been disabled by EventKillTimer() function.

Only one timer can be launched for each program. Each mql5 application and chart have their own queue of events where all newly arrived events are placed. If the queue already contains Timer event or this event is in the processing stage, then the new Timer event is not added to mql5 application queue.

EventSetTimer

The function indicates to the client terminal, that for this indicator or Expert Advisor, events from the timer must be generated with the specified periodicity.

```
bool  EventSetTimer(
   int  seconds      // number of seconds
   );
```

Parameters

seconds

[in] Number of seconds that determine the frequency of the timer event occurrence.

Return Value

In case of success returns true, otherwise false. In order to get an error code, the GetLastError() function should be called.

Note

Normally, this function must be called from the OnInit() function or from a class constructor. In order to handle events coming from the timer, the Expert Advisor must have the OnTimer() function.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql5 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the EventKillTimer() function.

For each program no more than one timer can be run. Each mql5 program and each chart has its own queue of events, in which all the newly received events are placed. If the Timer event is present in the queue or is being processed, the new Timer event will not be placed in the queue of the mql5 program.

EventKillTimer

Specifies the client terminal that is necessary to stop the generation of events from Timer.

```
void  EventKillTimer();
```

Return Value

No return value.

Note

Typically, this function must be called from a function OnDeinit(), if the EventSetTimer() function has been called from OnInit(). This function can also be called form the class destructor, if the EventSetTimer() function has been called in the constructor of this class.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql5 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the EventKillTimer() function

EventChartCustom

The function generates a custom event for the specified chart.

```
bool  EventChartCustom(
   long     chart_id,            // identifier of the event receiving chart
   ushort   custom_event_id,     // event identifier
   long     lparam,              // parameter of type long
   double   dparam,              // parameter of type double
   string   sparam               // string parameter of the event
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

custom_event_id

[in] ID of the user events. This identifier is automatically added to the value CHARTEVENT_CUSTOM and converted to the integer type.

lparam

[in] Event parameter of the long type passed to the OnChartEvent function.

dparam

[in] Event parameter of the double type passed to the OnChartEvent function.

sparam

[in] Event parameter of the string type passed to the OnChartEvent function. If the string is longer than 63 characters, it is truncated.

Return Value

Returns true if a custom event has been successfully placed in the events queue of the chart that receives the events. In case of failure, it returns false. Use GetLastError() to get an error code.

Note

An Expert Advisor or indicator attached to the specified chart handles the event using the function OnChartEvent(int event_id, long& lparam, double& dparam, string& sparam).

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through this parameters are listed in the below table.

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key	Repeat count (the number of times the keystroke is	The string value of a bit mask describing the

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
			repeated as a result of the user holding down the key)	status of keyboard buttons
Mouse event (if property CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate	the Y coordinate	The string value of a bit mask describing the status of mouse buttons
Event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	CHARTEVENT_OBJECT_CREATE	—	—	Name of the created graphical object
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	—	—	Name of the modified graphical object
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	—	—	Name of the deleted graphical object
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate	the Y coordinate	—
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate	the Y coordinate	Name of the graphical object, on which the event occurred
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	—	—	Name of the moved graphical object
Event of the finished text	CHARTEVENT_OBJECT_ENDEDIT	—	—	Name of the LabelEdit

Event	Value of the id parameter	Value of the lparam parameter	Value of the dparam parameter	Value of the sparam parameter
editing in the entry box of the LabelEdit graphical object				graphical object, in which text editing has completed
Event of changes on a chart	CHARTEVENT_CHART_CHANGE	—	—	—
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function	Value set by the EventChartCustom() function

Example:

```
//+------------------------------------------------------------------+
//|                                            ButtonClickExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Create a button to send custom events
   ObjectCreate(0,buttonID,OBJ_BUTTON,0,100,100);
   ObjectSetInteger(0,buttonID,OBJPROP_COLOR,clrWhite);
   ObjectSetInteger(0,buttonID,OBJPROP_BGCOLOR,clrGray);
   ObjectSetInteger(0,buttonID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_XSIZE,200);
   ObjectSetInteger(0,buttonID,OBJPROP_YSIZE,50);
   ObjectSetString(0,buttonID,OBJPROP_FONT,"Arial");
   ObjectSetString(0,buttonID,OBJPROP_TEXT,"Button");
   ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
   ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

//--- Create a label for displaying information
   ObjectCreate(0,labelID,OBJ_LABEL,0,100,100);
   ObjectSetInteger(0,labelID,OBJPROP_COLOR,clrRed);
   ObjectSetInteger(0,labelID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,labelID,OBJPROP_YDISTANCE,50);
   ObjectSetString(0,labelID,OBJPROP_FONT,"Trebuchet MS");
   ObjectSetString(0,labelID,OBJPROP_TEXT,"No information");
   ObjectSetInteger(0,labelID,OBJPROP_FONTSIZE,20);
   ObjectSetInteger(0,labelID,OBJPROP_SELECTABLE,0);

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ObjectDelete(0,buttonID);
   ObjectDelete(0,labelID);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Check the event by pressing a mouse button
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      string clickedChartObject=sparam;
      //--- If you click on the object with the name buttonID
      if(clickedChartObject==buttonID)
        {
         //--- State of the button - pressed or not
         bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
         //--- log a debug message
         Print("Button pressed = ",selected);
         int customEventID; // Number of the custom event to send
         string message;    // Message to be sent in the event
         //--- If the button is pressed
         if(selected)
```

```
           {
            message="Button pressed";
            customEventID=CHARTEVENT_CUSTOM+1;
           }
         else // Button is not pressed
           {
            message="Button in not pressed";
            customEventID=CHARTEVENT_CUSTOM+999;
           }
         //--- Send a custom event "our" chart
         EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
         ///--- Send a message to all open charts
         BroadcastEvent(ChartID(),0,"Broadcast Message");
         //--- Debug message
         Print("Sent an event with ID = ",customEventID);
        }
      ChartRedraw();// Forced redraw all chart objects
     }

//--- Check the event belongs to the user events
   if(id>CHARTEVENT_CUSTOM)
     {
      if(id==broadcastEventID)
        {
         Print("Got broadcast message from a chart with id = "+lparam);
        }
      else
        {
         //--- We read a text message in the event
         string info=sparam;
         Print("Handle the user event with the ID = ",id);
         //--- Display a message in a label
         ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
         ChartRedraw();// Forced redraw all chart objects
        }
     }
  }
//+------------------------------------------------------------------+
//| sends broadcast event to all open charts                         |
//+------------------------------------------------------------------+
void BroadcastEvent(long lparam,double dparam,string sparam)
  {
   int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
   long currChart=ChartFirst();
   int i=0;
   while(i<CHARTS_MAX)                 // We have certainly no more than CHARTS_MAX op
     {
      EventChartCustom(currChart,eventID,lparam,dparam,sparam);
      currChart=ChartNext(currChart); // We have received a new chart from the previo
      if(currChart==-1) break;        // Reached the end of the charts list
      i++;// Do not forget to increase the counter
     }
  }
//+------------------------------------------------------------------+
```

See also

Events of the client terminal, Event handler functions

Working with OpenCL

OpenCL programs are used for performing computations on video cards that support OpenCL 1.1 or higher. Modern video cards contain hundreds of small specialized processors that can simultaneously perform simple mathematical operations with incoming data streams. The OpenCL language organizes parallel computing and provides greater speed for a certain class of tasks.

In some graphic cards working with the double type numbers is disabled by default. This can lead to compilation error 5105. To enable support for the double type numbers, please add the following directive to your OpenCL program: #pragma OPENCL EXTENSION cl_khr_fp64 : enable. However if a graphic card doesn't support double, enabling this directive won't be of help.

It is recommended to write the source code for OpenCL in separate CL files, which can later be included in the MQL5 program using the resource variables.

Handling errors in OpenCL programs

To get information about the last error in an OpenCL program, use the CLGetInfoInteger and CLGetInfoStringfunctions that allow getting the error code and text description.

OpenCL last error code: To get the latest OpenCL error, call CLGetInfoInteger, while the*handle*parameter is ignored (can be set to zero). Description of errors: https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS.

For an unknown error code, the"unknown OpenCL error N" string is returned where N is an error code. Example:

```
//--- the first 'handle' parameter is ignored when getting the last error code
intcode= (int)CLGetInfoInteger(0,CL_LAST_ERROR);
```

Text description of the OpenCL error: To get the latest OpenCL error, call CLGetInfoString. The error code should be passed as the*handle*parameter.

Description of errors: https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS. If CL_LAST_ERROR is passed instead of the error code, then the function returns the description of the last error. For example:

```
//--- get the code of the last OpenCL error
int   code= (int)CLGetInfoInteger(0,CL_LAST_ERROR);
stringdesc;// to get an error text description

//--- use the error code to get an error text description
if(!CLGetInfoString(code,CL_ERROR_DESCRIPTION,desc))
 desc="cannot get OpenCL error description,"+ (string)GetLastError();
Print(desc);

//--- in order to get the description of the last OpenCL error without getting the co
if(!CLGetInfoString(CL_LAST_ERROR,CL_ERROR_DESCRIPTION,desc))
 desc="cannot get OpenCL error description,"+ (string)GetLastError();
Print(desc);;
```

So far, the name of the internal enumeration is given as an error description. You can find its decoding here: https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html#CL_SUCCESS. For example, the CL_INVALID_KERNEL_ARGS value means "Returned when enqueuing a kernel when some kernel arguments have not been set or are invalid."

Functions for running programs in OpenCL:

Function	Action
CLHandleType	Returns the type of an OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration
CLGetInfoInteger	Returns the value of an integer property for an OpenCL object or device
CLContextCreate	Creates an OpenCL context
CLContextFree	Removes an OpenCL context
CLGetDeviceInfo	Receives device property from OpenCL driver
CLProgramCreate	Creates an OpenCL program from a source code
CLProgramFree	Removes an OpenCL program
CLKernelCreate	Creates an OpenCL start function
CLKernelFree	Removes an OpenCL start function
CLSetKernelArg	Sets a parameter for the OpenCL function
CLSetKernelArgMem	Sets an OpenCL buffer as a parameter of the OpenCL function
CLSetKernelArgMemLocal	Sets the local buffer as an argument of the kernel function
CLBufferCreate	Creates an OpenCL buffer
CLBufferFree	Deletes an OpenCL buffer
CLBufferWrite	Writes an array into an OpenCL buffer
CLBufferRead	Reads an OpenCL buffer into an array
CLExecute	Runs an OpenCL program
CLExecutionStatus	Returns the OpenCL program execution status

See also

OpenCL, Resources

CLHandleType

Returns the type of an OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration.

```
ENUM_OPENCL_HANDLE_TYPE  CLHandleType(
   int  handle      // Handle of an OpenCL object
   );
```

Parameters

handle

[in] A handle to an OpenCL object: a context, a kernel or an OpenCL program.

Return Value

The type of the OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration.

ENUM_OPENCL_HANDLE_TYPE

Identifier	Description
OPENCL_INVALID	Incorrect handle
OPENCL_CONTEXT	A handle of the OpenCL context
OPENCL_PROGRAM	A handle of the OpenCL program
OPENCL_KERNEL	A handle of the OpenCL kernel
OPENCL_BUFFER	A handle of the OpenCL buffer

CLGetInfoInteger

Returns the value of an integer property for an OpenCL object or device.

```
long  CLGetInfoInteger(
   int  handle,                          // The handle of the OpenCL object or the nu
   ENUM_OPENCL_PROPERTY_INTEGER  prop      // Requested property
   );
```

Parameters

handle

[in] A handle to the OpenCL object or number of the OpenCL device. Numbering of OpenCL devices starts with zero.

prop

[in] The type of a requested property from the ENUM_OPENCL_PROPERTY_INTEGER enumeration, the value of which you want to obtain.

Return Value

The value of the property if successful or -1 in case of an error. For information about the error, use the GetLastError() function.

ENUM_OPENCL_PROPERTY_INTEGER

Identifier	Description	Type
CL_DEVICE_COUNT	The number of devices with OpenCL support. This property does not require specification of the first parameter, i.e. you can pass a zero value for the *handle* parameter.	int
CL_DEVICE_TYPE	Type of device	ENUM_CL_DEVICE_TYPE
CL_DEVICE_VENDOR_ID	Unique vendor identifier	uint
CL_DEVICE_MAX_COMPUTE_UNITS	Number of parallel calculated tasks in OpenCL device. One working group solves one computational task. The lowest value is 1	uint
CL_DEVICE_MAX_CLOCK_FREQUENCY	Highest set frequency of the device in MHz.	uint
CL_DEVICE_GLOBAL_MEM_SIZE	Size of the global memory of the device in bytes	ulong
CL_DEVICE_LOCAL_MEM_SIZE	Size of the processed data (scene) local memory in bytes	uint

Identifier	Description	Type
CL_BUFFER_SIZE	Actual size of the OpenCL buffer in bytes	ulong
CL_DEVICE_MAX_WORK_GROUP_SIZE	The total number of the local working groups available for an OpenCL device.	ulong
CL_KERNEL_WORK_GROUP_SIZE	The total number of the local working groups available for an OpenCL program.	ulong
CL_KERNEL_LOCAL_MEM_SIZE	Size of the local memory (in bytes) used by an OpenCL program for solving all parallel tasks in a group. Use CL_DEVICE_LOCAL_MEM_SIZE to receive the maximum available value	ulong
CL_KERNEL_PRIVATE_MEM_SIZE	The minimum size of the private memory (in bytes) used by each task in the OpenCL program kernel.	ulong
CL_LAST_ERROR	The value of the last OpenCL error	int

The ENUM_CL_DEVICE_TYPE enumeration contains possible types of devices supporting OpenCL. You can receive the type of device by its number or the handle of the OpenCL object by calling CLGetInfoInteger(handle_or_deviceN, CL_DEVICE_TYPE).

ENUM_CL_DEVICE_TYPE

Identifier	Description
CL_DEVICE_ACCELERATOR	Dedicated OpenCL accelerators (for example, the IBM CELL Blade). These devices communicate with the host processor using a peripheral interconnect such as PCIe.
CL_DEVICE_CPU	An OpenCL device that is the host processor. The host processor runs the OpenCL implementations and is a single or multi-core CPU.
CL_DEVICE_GPU	An OpenCL device that is a GPU.
CL_DEVICE_DEFAULT	The default OpenCL device in the system. The default device cannot be a CL_DEVICE_TYPE_CUSTOM device.

Identifier	Description
CL_DEVICE_CUSTOM	Dedicated accelerators that do not support programs written in OpenCL C.

Example:

```
void OnStart()
  {
   int cl_ctx;
//--- initialize OpenCL context
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
//--- Display general information about OpenCL device
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY)," MHz"
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE)," by
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE)," byte
//--- free OpenCL context
   CLContextFree(cl_ctx);
  }
```

CLGetInfoString

Returns string value of a property for OpenCL object or device.

```
bool  CLGetInfoString(
   int  handle,                          // OpenCL object handle or OpenCL device nu
   ENUM_OPENCL_PROPERTY_STRING  prop,    // requested property
   string&  value                        // referenced string
   );
```

Parameters

handle

[in] OpenCL object handle or OpenCL device number. The numbering of OpenCL devices starts with zero.

prop

[in] Type of requested property from ENUM_OPENCL_PROPERTY_STRING enumeration, the value of which should be obtained.

value

[out] String for receiving the property value.

Return Value

true if successful, otherwise false. For information about the error, use the GetLastError() function.

ENUM_OPENCL_PROPERTY_STRING

Identifier	Description
CL_PLATFORM_PROFILE	CL_PLATFORM_PROFILE - OpenCL Profile. Profile name may be one of the following values: • FULL_PROFILE - implementation supports OpenCL (functionality is defined as the part of the kernel specification without requiring additional extensions for OpenCL support); • EMBEDDED_PROFILE - implementation supports OpenCL as a supplement. Amended profile is defined as a subset for each OpenCL version.
CL_PLATFORM_VERSION	OpenCL version
CL_PLATFORM_VENDOR	Platform vendor name
CL_PLATFORM_EXTENSIONS	List of extensions supported by the platform. Extension names should be supported by all devices related to this platform
CL_DEVICE_NAME	Device name
CL_DEVICE_VENDOR	Vendor name

Identifier	Description
CL_DRIVER_VERSION	OpenCL driver version in major_number.minor_number format
CL_DEVICE_PROFILE	OpenCL device profile. Profile name may be one of the following values: • FULL_PROFILE - implementation supports OpenCL (functionality is defined as the part of the kernel specification without requiring additional extensions for OpenCL support); • EMBEDDED_PROFILE - implementation supports OpenCL as a supplement. Amended profile is defined as a subset for each OpenCL version.
CL_DEVICE_VERSION	OpenCL version in "OpenCL<space><major_version.minor_version><space><vendor-specific information>" format
CL_DEVICE_EXTENSIONS	List of extensions supported by the device. The list may contain extensions supported by the vendor, as well as one or more approved names: cl_khr_int64_base_atomics cl_khr_int64_extended_atomics cl_khr_fp16 cl_khr_gl_sharing cl_khr_gl_event cl_khr_d3d10_sharing cl_khr_dx9_media_sharing cl_khr_d3d11_sharing
CL_DEVICE_BUILT_IN_KERNELS	The list of supported kernels separated by ";".
CL_DEVICE_OPENCL_C_VERSION	The maximum version supported by the compiler for this device. Version format: "OpenCL<space>C<space><major_version.minor_version><space><vendor-specific information> "
CL_ERROR_DESCRIPTION	Text description of an OpenCL error

Example:

```
void OnStart()
  {
   int cl_ctx;
   string str;
//--- initialize OpenCL context
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
```

```
//--- Display information about the platform
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_NAME,str))
      Print("OpenCL platform name: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VENDOR,str))
      Print("OpenCL platform vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VERSION,str))
      Print("OpenCL platform ver: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_PROFILE,str))
      Print("OpenCL platform profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_EXTENSIONS,str))
      Print("OpenCL platform ext: ",str);
//--- Display information about the device
   if(CLGetInfoString(cl_ctx,CL_DEVICE_NAME,str))
      Print("OpenCL device name: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_PROFILE,str))
      Print("OpenCL device profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_BUILT_IN_KERNELS,str))
      Print("OpenCL device kernels: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_EXTENSIONS,str))
      Print("OpenCL device ext: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VENDOR,str))
      Print("OpenCL device vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VERSION,str))
      Print("OpenCL device ver: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_OPENCL_C_VERSION,str))
      Print("OpenCL open c ver: ",str);
//--- Display general information about the OpenCL device
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY));
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE));
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE));
//--- free OpenCL context
   CLContextFree(cl_ctx);
  }
```

CLContextCreate

Creates an OpenCL context and returns its handle.

```
int  CLContextCreate(
   int  device=CL_USE_ANY     // Serial number of the OpenCL device or macro
   );
```

Parameter

device

[in] The ordinal number of the OpenCL-device in the system. Instead of a specific number, you can specify one of the following values:

- CL_USE_ANY – any available device with OpenCL support is allowed;
- CL_USE_CPU_ONLY – only OpenCL emulation on CPU is allowed;
- CL_USE_GPU_ONLY – OpenCL emulation is prohibited and only specialized devices with OpenCL support (video cards) can be used;
- CL_USE_GPU_DOUBLE_ONLY – only the GPUs that support type double are allowed.

Return Value

A handle to the OpenCL context if successful, otherwise -1. For information about the error, use the GetLastError() function.

CLContextFree

Removes an OpenCL context.

```
void  CLContextFree(
   int  context      // Handle to an OpenCL context
   );
```

Parameters

context

[in] Handle of the OpenCL context.

Return Value

None. In the case of an internal error the value of _LastError changes. For information about the error, use the GetLastError() function.

CLGetDeviceInfo

The function receives device property from OpenCL driver.

```
bool  CLGetDeviceInfo(
   int      handle,           // OpenCL device handle
   int      property_id,      // requested property ID
   uchar&   data[],           // array for receiving data
   uint&    size              // shift in the array elements, default value is 0
   );
```

Parameters

handle

[in] OpenCL device index or OpenCL handle created by CLContextCreate() function.

property_id

[in] ID of the OpenCL device property that should be received. The values can be of one of the predetermined ones listed in the table below.

data[]

[out] The array for receiving data on the requested property.

size

[out] Size of the received data in the array *data[]*.

Return Value

true if successful, otherwise false. For information about the error, use the GetLastError() function.

Note

For one-dimensional arrays, the number of the element, from which data reading for OpenCL buffer starts, is calculated considering AS_SERIES flag.

The list of available IDs of OpenCL device properties

Exact description of the property and its functions can be found at the official OpenCL web site.

Identifier	Value
CL_DEVICE_TYPE	0x1000
CL_DEVICE_VENDOR_ID	0x1001
CL_DEVICE_MAX_COMPUTE_UNITS	0x1002
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	0x1003
CL_DEVICE_MAX_WORK_GROUP_SIZE	0x1004
CL_DEVICE_MAX_WORK_ITEM_SIZES	0x1005
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR	0x1006

Identifier	Value
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT	0x1007
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT	0x1008
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG	0x1009
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT	0x100A
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE	0x100B
CL_DEVICE_MAX_CLOCK_FREQUENCY	0x100C
CL_DEVICE_ADDRESS_BITS	0x100D
CL_DEVICE_MAX_READ_IMAGE_ARGS	0x100E
CL_DEVICE_MAX_WRITE_IMAGE_ARGS	0x100F
CL_DEVICE_MAX_MEM_ALLOC_SIZE	0x1010
CL_DEVICE_IMAGE2D_MAX_WIDTH	0x1011
CL_DEVICE_IMAGE2D_MAX_HEIGHT	0x1012
CL_DEVICE_IMAGE3D_MAX_WIDTH	0x1013
CL_DEVICE_IMAGE3D_MAX_HEIGHT	0x1014
CL_DEVICE_IMAGE3D_MAX_DEPTH	0x1015
CL_DEVICE_IMAGE_SUPPORT	0x1016
CL_DEVICE_MAX_PARAMETER_SIZE	0x1017
CL_DEVICE_MAX_SAMPLERS	0x1018
CL_DEVICE_MEM_BASE_ADDR_ALIGN	0x1019
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE	0x101A
CL_DEVICE_SINGLE_FP_CONFIG	0x101B
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE	0x101C
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE	0x101D
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE	0x101E
CL_DEVICE_GLOBAL_MEM_SIZE	0x101F
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE	0x1020
CL_DEVICE_MAX_CONSTANT_ARGS	0x1021
CL_DEVICE_LOCAL_MEM_TYPE	0x1022

Identifier	Value
CL_DEVICE_LOCAL_MEM_SIZE	0x1023
CL_DEVICE_ERROR_CORRECTION_SUPPORT	0x1024
CL_DEVICE_PROFILING_TIMER_RESOLUTION	0x1025
CL_DEVICE_ENDIAN_LITTLE	0x1026
CL_DEVICE_AVAILABLE	0x1027
CL_DEVICE_COMPILER_AVAILABLE	0x1028
CL_DEVICE_EXECUTION_CAPABILITIES	0x1029
CL_DEVICE_QUEUE_PROPERTIES	0x102A
CL_DEVICE_NAME	0x102B
CL_DEVICE_VENDOR	0x102C
CL_DRIVER_VERSION	0x102D
CL_DEVICE_PROFILE	0x102E
CL_DEVICE_VERSION	0x102F
CL_DEVICE_EXTENSIONS	0x1030
CL_DEVICE_PLATFORM	0x1031
CL_DEVICE_DOUBLE_FP_CONFIG	0x1032
CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF	0x1034
CL_DEVICE_HOST_UNIFIED_MEMORY	0x1035
CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR	0x1036
CL_DEVICE_NATIVE_VECTOR_WIDTH_SHORT	0x1037
CL_DEVICE_NATIVE_VECTOR_WIDTH_INT	0x1038
CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG	0x1039
CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT	0x103A
CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE	0x103B
CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF	0x103C
CL_DEVICE_OPENCL_C_VERSION	0x103D
CL_DEVICE_LINKER_AVAILABLE	0x103E
CL_DEVICE_BUILT_IN_KERNELS	0x103F
CL_DEVICE_IMAGE_MAX_BUFFER_SIZE	0x1040
CL_DEVICE_IMAGE_MAX_ARRAY_SIZE	0x1041

Identifier	Value
CL_DEVICE_PARENT_DEVICE	0x1042
CL_DEVICE_PARTITION_MAX_SUB_DEVICES	0x1043
CL_DEVICE_PARTITION_PROPERTIES	0x1044
CL_DEVICE_PARTITION_AFFINITY_DOMAIN	0x1045
CL_DEVICE_PARTITION_TYPE	0x1046
CL_DEVICE_REFERENCE_COUNT	0x1047
CL_DEVICE_PREFERRED_INTEROP_USER_SYNC	0x1048
CL_DEVICE_PRINTF_BUFFER_SIZE	0x1049
CL_DEVICE_IMAGE_PITCH_ALIGNMENT	0x104A
CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT	0x104B

Example:

```
void OnStart()
  {
//---
   int dCount= CLGetInfoInteger(0,CL_DEVICE_COUNT);
   for(int i = 0; i<dCount; i++)
     {
      int clCtx=CLContextCreate(i);
      if(clCtx == -1)
         Print("ERROR in CLContextCreate");
      string device;
      CLGetInfoString(clCtx,CL_DEVICE_NAME,device);
      Print(i,": ",device);
      uchar data[1024];
      uint size;
      CLGetDeviceInfo(clCtx,CL_DEVICE_VENDOR,data,size);
      Print("size = ",size);
      string str=CharArrayToString(data);
      Print(str);
     }
  }
//--- example of entries in Experts journal
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      2: Advanced Micro Devices, Inc.
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      size = 32
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      Tahiti
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      Intel(R) Corporation
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      size = 21
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      1:         Intel(R) Core(TM) i7-37
//  2013.07.24 10:50:48     opencl (EURUSD,H1)      NVIDIA Corporation
//   2013.07.24 10:50:48      opencl (EURUSD,H1)       size = 19
//   2013.07.24 10:50:48      opencl (EURUSD,H1)       0: GeForce GTX 580
```

CLProgramCreate

Creates an OpenCL program from a source code.

```
int  CLProgramCreate(
   int              context,       // Handle to an OpenCL context
   const string     source         // Source code
   );
```

An overloaded function version creates an OpenCL program and writes compiler messages into the passed string.

```
int  CLProgramCreate(
   int              context,       // Handle to an OpenCL context
   const string     source,        // Source code
   string          &build_log      // A string for receiving the compilation log
   );
```

Parameters

context

[in] Handle of the OpenCL context.

source

[in] String with the source code of the OpenCL program.

&build_log

[in] A string for receiving the OpenCL compiler messages.

Return Value

A handle to an OpenCL object if successful. In case of error -1 is returned. For information about the error, use the GetLastError() function.

Note

At the moment, the following error codes are used:

- ERR_OPENCL_INVALID_HANDLE – invalid handle to the context OpenCL.
- ERR_INVALID_PARAMETER – invalid string parameter.
- ERR_NOT_ENOUGH_MEMORY – not enough memory to complete operation.
- ERR_OPENCL_PROGRAM_CREATE – internal error of OpenCL or compilation error.

In some graphic cards working with the double type numbers is disabled by default. This can lead to compilation error 5105. To enable support for the double type numbers, please add the following directive to your OpenCL program: #pragma OPENCL EXTENSION cl_khr_fp64 : enable. However if a graphic card doesn't support double, enabling this directive won't be of help.

Example:

```
//+------------------------------------------------------------------+
//| OpenCL kernel                                                    |
//+------------------------------------------------------------------+
const string
```

```
cl_src=
       //--- by default some GPU doesn't support doubles
       //--- cl_khr_fp64 directive is used to enable work with doubles
       "#pragma OPENCL EXTENSION cl_khr_fp64 : enable      \r\n"
       //--- OpenCL kernel function
       "__kernel void Test_GPU(__global double *data,      \r\n"
       "                       const    int N,             \r\n"
       "                       const    int total_arrays)  \r\n"
       "  {                                                \r\n"
       "   uint kernel_index=get_global_id(0);             \r\n"
       "   if (kernel_index>total_arrays) return;          \r\n"
       "   uint local_start_offset=kernel_index*N;         \r\n"
       "   for(int i=0; i<N; i++)                          \r\n"
       "     {                                             \r\n"
       "       data[i+local_start_offset] *= 2.0;          \r\n"
       "     }                                             \r\n"
       "  }                                                \r\n";
//+------------------------------------------------------------------+
//| Test_CPU                                                         |
//+------------------------------------------------------------------+
bool Test_CPU(double &data[],const int N,const int id,const int total_arrays)
  {
//--- check array size
   if(ArraySize(data)==0) return(false);
//--- check array index
   if(id>total_arrays) return(false);
//--- calculate local offset for array with index id
   int local_start_offset=id*N;
//--- multiply elements by 2
   for(int i=0; i<N; i++)
     {
      data[i+local_start_offset]*=2.0;
     }
   return true;
  }
//---
#define ARRAY_SIZE   100  // size of the array
#define TOTAL_ARRAYS 5    // total arrays
//--- OpenCL handles
int cl_ctx;  // OpenCL context handle
int cl_prg;  // OpenCL program handle
int cl_krn;  // OpenCL kernel handle
int cl_mem;  // OpenCL buffer handle
//---
double DataArray1[]; // data array for CPU calculation
double DataArray2[]; // data array for GPU calculation
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart()
  {
//--- initialize OpenCL objects
//--- create OpenCL context
   if((cl_ctx=CLContextCreate())==INVALID_HANDLE)
     {
      Print("OpenCL not found. Error=",GetLastError());
      return(1);
     }
//--- create OpenCL program
   if((cl_prg=CLProgramCreate(cl_ctx,cl_src))==INVALID_HANDLE)
     {
      CLContextFree(cl_ctx);
      Print("OpenCL program create failed. Error=",GetLastError());
      return(1);
     }
//--- create OpenCL kernel
   if((cl_krn=CLKernelCreate(cl_prg,"Test_GPU"))==INVALID_HANDLE)
     {
      CLProgramFree(cl_prg);
      CLContextFree(cl_ctx);
      Print("OpenCL kernel create failed. Error=",GetLastError());
      return(1);
     }
//--- create OpenCL buffer
   if((cl_mem=CLBufferCreate(cl_ctx,ARRAY_SIZE*TOTAL_ARRAYS*sizeof(double),CL_MEM_REA
     {
      CLKernelFree(cl_krn);
      CLProgramFree(cl_prg);
      CLContextFree(cl_ctx);
      Print("OpenCL buffer create failed. Error=",GetLastError());
      return(1);
     }
//--- set OpenCL kernel constant parameters
   CLSetKernelArgMem(cl_krn,0,cl_mem);
   CLSetKernelArg(cl_krn,1,ARRAY_SIZE);
   CLSetKernelArg(cl_krn,2,TOTAL_ARRAYS);
//--- prepare data arrays
   ArrayResize(DataArray1,ARRAY_SIZE*TOTAL_ARRAYS);
   ArrayResize(DataArray2,ARRAY_SIZE*TOTAL_ARRAYS);
//--- fill arrays with data
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- calculate local start offset for jth array
      uint local_offset=j*ARRAY_SIZE;
      //--- prepare array with index j
      for(int i=0; i<ARRAY_SIZE; i++)
        {
         //--- fill arrays with function MathCos(i+j);
         DataArray1[i+local_offset]=MathCos(i+j);
         DataArray2[i+local_offset]=MathCos(i+j);
        }
     };
//--- test CPU calculation
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- calculation of the array with index j
      Test_CPU(DataArray1,ARRAY_SIZE,j,TOTAL_ARRAYS);
     }
//--- prepare CLExecute params
   uint  offset[]={0};
//--- global work size
   uint  work[]={TOTAL_ARRAYS};
//--- write data to OpenCL buffer
   CLBufferWrite(cl_mem,DataArray2);
//--- execute OpenCL kernel
   CLExecute(cl_krn,1,offset,work);
//--- read data from OpenCL buffer
   CLBufferRead(cl_mem,DataArray2);
//--- total error
   double total_error=0;
//--- compare results and calculate error
   for(int j=0; j<TOTAL_ARRAYS; j++)
     {
      //--- calculate local offset for jth array
      uint local_offset=j*ARRAY_SIZE;
      //--- compare the results
      for(int i=0; i<ARRAY_SIZE; i++)
        {
         double v1=DataArray1[i+local_offset];
         double v2=DataArray2[i+local_offset];
         double delta=MathAbs(v2-v1);
         total_error+=delta;
         //--- show first and last arrays
         if((j==0) || (j==TOTAL_ARRAYS-1))
            PrintFormat("array %d of %d, element [%d]:  %f, %f, [error]=%f",j+1,TOTAL_
        }
     }
   PrintFormat("Total error: %f",total_error);
//--- delete OpenCL objects
//--- free OpenCL buffer
   CLBufferFree(cl_mem);
//--- free OpenCL kernel
   CLKernelFree(cl_krn);
//--- free OpenCL program
   CLProgramFree(cl_prg);
//--- free OpenCL context
   CLContextFree(cl_ctx);
```

```
//---
   return(0);
  }
```

CLProgramFree

Removes an OpenCL program.

```
void  CLProgramFree(
   int  program      // Handle to an OpenCL object
   );
```

Parameters

program

[in] Handle of the OpenCL object.

Return Value

None. In the case of an internal error the value of _LastError changes. For information about the error, use the GetLastError() function.

CLKernelCreate

Creates the OpenCL program kernel and returns its handle.

```
int  CLKernelCreate(
   int             program,         // Handle to an OpenCL object
   const string  kernel_name      // Kernel name
   );
```

Parameters

program

[in] Handle to an object of the OpenCL program.

kernel_name

[in] The name of the kernel function in the appropriate OpenCL program, in which execution begins.

Return Value

A handle to an OpenCL object if successful. In case of error -1 is returned. For information about the error, use the GetLastError() function.

Note

At the moment, the following error codes are used:

- ERR_OPENCL_INVALID_HANDLE - invalid handle to OpenCL program.
- ERR_INVALID_PARAMETER - invalid string parameter.
- ERR_OPENCL_TOO_LONG_KERNEL_NAME - kernel name contains more than 127 characters.
- ERR_OPENCL_KERNEL_CREATE - internal error occurred while creating an OpenCL object.

CLKernelFree

Removes an OpenCL start function.

```
void  CLKernelFree(
   int  kernel      // Handle to the kernel of an OpenCL program
   );
```

Parameters

kernel_name

[in] Handle of the kernel object.

Return Value

None. In the case of an internal error the value of _LastError changes. For information about the error, use the GetLastError() function.

CLSetKernelArg

Sets a parameter for the OpenCL function.

```
bool  CLSetKernelArg(
   int    kernel,         // Handle to the kernel of an OpenCL program
   uint   arg_index,      // The number of the argument of the OpenCL function
   void   arg_value       // Source code
   );
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

arg_value

[in] The value of the function argument.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the GetLastError() function.

Note

At the moment, the following error codes are used:

- ERR_INVALID_PARAMETER,
- ERR_OPENCL_INVALID_HANDLE - invalid handle to the OpenCL kernel.
- ERR_OPENCL_SET_KERNEL_PARAMETER - internal error of OpenCL.

CLSetKernelArgMem

Sets an OpenCL buffer as a parameter of the OpenCL function.

```
bool  CLSetKernelArgMem(
   int    kernel,           // Handle to the kernel of an OpenCL program
   uint   arg_index,        // The number of the argument of the OpenCL function
   int    cl_mem_handle     // Handle to OpenCL buffer
   );
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

cl_mem_handle

[in] A handle to an OpenCL buffer.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the GetLastError() function.

CLSetKernelArgMemLocal

Sets the local buffer as an argument of the kernel function.

```
bool  CLSetKernelArgMemLocal(
   int     kernel,            // handle to a kernel of an OpenCL program
   uint    arg_index,         // number of the OpenCL function argument
   ulong   local_mem_size     // buffer size
   );
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

arg_index

[in] The number of the function argument, numbering starts with zero.

local_mem_size

[in] Buffer size in bytes.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the GetLastError() function.

CLBufferCreate

Creates an OpenCL buffer and returns its handle.

```
int  CLBufferCreate(
   int    context,      // Handle to an OpenCL context
   uint   size,         // Buffer size
   uint   flags         // Flags combination which specify properties of OpenCL buffer
   );
```

Parameters

context

[in] A handle to context OpenCL.

size

[in] Buffer size in bytes.

flags

[in] Buffer properties that are set using a combination of flags: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY, CL_MEM_ALLOC_HOST_PTR.

Return Value

A handle to an OpenCL buffer if successful. In case of error -1 is returned. For information about the error, use the GetLastError() function.

Note

At the moment, the following error codes are used:

- ERR_OPENCL_INVALID_HANDLE - invalid handle to OpenCL context.
- ERR_NOT_ENOUGH_MEMORY - insufficient memory.
- ERR_OPENCL_BUFFER_CREATE - internal error creating buffers.

CLBufferFree

Deletes an OpenCL buffer.

```
void  CLBufferFree(
   int   buffer      // Handle to an OpenCL buffer
   );
```

Parameters

buffer

[in] A handle to an OpenCL buffer.

Return Value

None. In the case of an internal error the value of _LastError changes. For information about the error, use the GetLastError() function.

CLBufferWrite

Writes into the OpenCL buffer and returns the number of written elements.

```
uint  CLBufferWrite(
   int            buffer,                    // A handle to the OpenCL buffer
   const void&    data[],                    // An array of values
   uint           buffer_offset=0,           // An offset in the OpenCL buffer in bytes,
   uint           data_offset=0,             // An offset in the array in elements, 0 by
   uint           data_count=WHOLE_ARRAY     // The number of values from the array for
   );
```

There are also versions for handling matrices and vectors.

Writes the values from the matrix to the buffer and returns true if successful.

```
uint  CLBufferWrite(
   int             buffer,                    // a handle to the OpenCL buffer
   uint            buffer_offset,             // an offset in the OpenCL buffer in bytes
   matrix<T>       &mat                       // the values matrix for writing to the bu
   );
```

Writes the values from the vector to the buffer and returns true if successful.

```
uint  CLBufferWrite(
   int             buffer,                    // a handle to the OpenCL buffer
   uint            buffer_offset,             // an offset in the OpenCL buffer in bytes
   vector<T>       &vec                       // the values vector for writing to the bu
   );
```

Parameters

buffer

[in] A handle of the OpenCL buffer.

data[]

[in] An array of values that should be written in the OpenCL buffer. Passed by reference.

buffer_offset

[in] An offset in the OpenCL buffer in bytes, from which writing begins. By default, writing start with the very beginning of the buffer.

data_offset

[in] The index of the first array element, starting from which values from the array are written in the OpenCL buffer. By default, values from the very beginning of the array are taken.

data_count

[in] The number of values that should be written. All the values of the array, by default.

mat

[out] The matrix for reading data from the buffer can be any of the three types — matrix, matrixf or matrixc.

vec

[out] The vector for reading data from the buffer can be of any of the three types – vector, vectorf or vectorc.

Return Value

The number of written elements. 0 is returned in case of an error. For information about the error, use the GetLastError() function.

true if a matrix or a vector is handled successfully, otherwise false.

Note

For one-dimensional arrays, the number of the element, with which reading of data for writing into an OpenCL buffer begins, is calculated taking into account the AS_SERIES flags.

An array of two or more dimensions is presented as one-dimensional. In this case, *data_offset* is the number of elements that should be skipped in the presentation, not the number of elements in the first dimension.

Example of matrix multiplication using the MatMul method and parallel computing in OpenCL

```
#define M       3000      // the number of rows in the first matrix
#define K       2000      // the number of columns in the first matrix is equal to the
#define N       3000      // the number of columns in the second matrix

//+------------------------------------------------------------------+
const string clSrc=
  "#define N     "+IntegerToString(N)+"                              \r\n"
  "#define K     "+IntegerToString(K)+"                              \r\n"
  "                                                                  \r\n"
  "__kernel void matricesMul( __global float *in1,                   \r\n"
  "                           __global float *in2,                   \r\n"
  "                           __global float *out  )                 \r\n"
  "{                                                                 \r\n"
  "  int m = get_global_id( 0 );                                     \r\n"
  "  int n = get_global_id( 1 );                                     \r\n"
  "  float sum = 0.0;                                                \r\n"
  "  for( int k = 0; k < K; k ++ )                                   \r\n"
  "     sum += in1[ m * K + k ] * in2[ k * N + n ];                  \r\n"
  "  out[ m * N + n ] = sum;                                         \r\n"
  "}                                                                 \r\n";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
//--- initialize the random number generator
  MathSrand((int)TimeCurrent());
//--- fill in the matrices of a given size with random values
```

```
   matrixf mat1(M, K, MatrixRandom) ;    // first matrix
   matrixf mat2(K, N, MatrixRandom);     // second matrix

//--- calculate the product of matrices using naive method
   uint start=GetTickCount();
   matrixf matrix_naive=matrixf::Zeros(M, N);// the result of multiplying two matrices
   for(int m=0; m<M; m++)
     for(int k=0; k<K; k++)
       for(int n=0; n<N; n++)
         matrix_naive[m][n]+=mat1[m][k]*mat2[k][n];
   uint time_naive=GetTickCount()-start;

//--- calculate the product of matrices via MatMull
   start=GetTickCount();
   matrixf matrix_matmul=mat1.MatMul(mat2);
   uint time_matmul=GetTickCount()-start;

//--- calculate the product of matrices in OpenCL
   matrixf matrix_opencl=matrixf::Zeros(M, N);
   int cl_ctx;              // context handle
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
    {
     Print("OpenCL not found, leaving");
     return;
    }
   int cl_prg;              // program handle
   int cl_krn;              // kernel handle
   int cl_mem_in1;          // first (input) buffer handle
   int cl_mem_in2;          // second (input) buffer handle
   int cl_mem_out;          // third (output) buffer handle
//--- create the program and the kernel
   cl_prg = CLProgramCreate(cl_ctx, clSrc);
   cl_krn = CLKernelCreate(cl_prg, "matricesMul");
//--- create all three buffers for three matrices
   cl_mem_in1=CLBufferCreate(cl_ctx, M*K*sizeof(float), CL_MEM_READ_WRITE);
   cl_mem_in2=CLBufferCreate(cl_ctx, K*N*sizeof(float), CL_MEM_READ_WRITE);
//--- third matrix - output
   cl_mem_out=CLBufferCreate(cl_ctx, M*N*sizeof(float), CL_MEM_READ_WRITE);
//--- set the kernel arguments
   CLSetKernelArgMem(cl_krn, 0, cl_mem_in1);
   CLSetKernelArgMem(cl_krn, 1, cl_mem_in2);
   CLSetKernelArgMem(cl_krn, 2, cl_mem_out);
//--- write matrices to the device buffers
   CLBufferWrite(cl_mem_in1, 0, mat1);
   CLBufferWrite(cl_mem_in2, 0, mat2);
   CLBufferWrite(cl_mem_out, 0, matrix_opencl);
//--- OpenCL code execution time start
   start=GetTickCount();
//--- set the parameters of the task working area and execute the OpenCL program
```

```
  uint  offs[2] = {0, 0};
  uint works[2] = {M, N};
  start=GetTickCount();
  bool ex=CLExecute(cl_krn, 2, offs, works);
//--- calculate the result to the matrix
  if(CLBufferRead(cl_mem_out, 0, matrix_opencl))
    PrintFormat("[%d x %d] matrix read: ", matrix_opencl.Rows(), matrix_opencl.Cols())
   else
      Print("CLBufferRead(cl_mem_out, 0, matrix_opencl failed. Error ",GetLastError())
  uint time_opencl=GetTickCount()-start;
  Print("Compare calculation time using each method");
  PrintFormat("Naive product time = %d ms",time_naive);
  PrintFormat("MatMul product time = %d ms",time_matmul);
  PrintFormat("OpenCl product time = %d ms",time_opencl);
//--- release all OpenCL contexts
  CLFreeAll(cl_ctx, cl_prg, cl_krn, cl_mem_in1, cl_mem_in2, cl_mem_out);

//--- compare all obtained result matrices with each other
  Print("How many discrepancy errors are there between result matrices?");
  ulong errors=matrix_naive.Compare(matrix_matmul,(float)1e-12);
  Print("matrix_direct.Compare(matrix_matmul,1e-12)=",errors);
  errors=matrix_matmul.Compare(matrix_opencl,float(1e-12));
  Print("matrix_matmul.Compare(matrix_opencl,1e-12)=",errors);
/*
  Result:

   [3000 x 3000] matrix read:
  Compare the time of calculation with each method
   Naive product time = 54750 ms
   MatMul product time = 4578 ms
   OpenCl product time = 922 ms
   How many discrepancy errors are there between result matrices?
   matrix_direct.Compare(matrix_matmul,1e-12)=0
   matrix_matmul.Compare(matrix_opencl,1e-12)=0
*/
 }
//+------------------------------------------------------------------+
//| Fills the matrix with random values                              |
//+------------------------------------------------------------------+
void MatrixRandom(matrixf& m)
 {
  for(ulong r=0; r<m.Rows(); r++)
   {
    for(ulong c=0; c<m.Cols(); c++)
     {
      m[r][c]=(float)((MathRand()-16383.5)/32767.);
     }
   }
 }
```

```
//+------------------------------------------------------------------+
//| Release all OpenCL contexts                                      |
//+------------------------------------------------------------------+
void CLFreeAll(int cl_ctx, int cl_prg, int cl_krn,
               int cl_mem_in1, int cl_mem_in2, int cl_mem_out)
 {
//--- delete all contexts created by OpenCL in reverse order
  CLBufferFree(cl_mem_in1);
  CLBufferFree(cl_mem_in2);
  CLBufferFree(cl_mem_out);
  CLKernelFree(cl_krn);
  CLProgramFree(cl_prg);
  CLContextFree(cl_ctx);
 }
```

CLBufferRead

Reads an OpenCL buffer into an array and returns the number of read elements.

```
uint  CLBufferRead(
   int          buffer,                    // A handle to the OpenCL buffer
   const void&  data[],                    // An array of values
   uint         buffer_offset=0,           // An offset in the OpenCL buffer in bytes,
   uint         data_offset=0,             // An offset in the array in elements, 0 by
   uint         data_count=WHOLE_ARRAY     // The number of values from the buffer for
   );
```

There are also versions for handling matrices and vectors.

Reads the OpenCL buffer to the matrix and returns true if successful.

```
uint  CLBufferRead(
   int            buffer,                    // a handle to the OpenCL buffer
   uint           buffer_offset,             // an offset in the OpenCL buffer in bytes
   const matrix&  mat,                       // the matrix for receiving the values fro
   ulong          rows=-1,                   // the number of rows in the matrix
   ulong          cols=-1                    // the number of columns in the matrix
   );
```

Reads the OpenCL buffer to the vector and returns true if successful.

```
uint  CLBufferRead(
   int            buffer,                    // a handle to the OpenCL buffer
   uint           buffer_offset,             // an offset in the OpenCL buffer in bytes
   const vector&  vec,                       // the vector for receiving the values fro
   ulong          size-1,                    // vector length
   );
```

Parameters

buffer

[in] A handle of the OpenCL buffer.

data[]

[in] An array for receiving values from the OpenCL buffer. Passed by reference.

buffer_offset

[in] An offset in the OpenCL buffer in bytes, from which reading begins. By default, reading start with the very beginning of the buffer.

data_offset

[in] The index of the first array element for writing the values of the OpenCL buffer. By default, writing of the read values into an array starts from the zero index.

data_count

[in] The number of values that should be read. The whole OpenCL buffer is read by default.

mat

[out] The matrix for reading data from the buffer can be any of the three types — matrix, matrixf or matrixc.

vec

[out] The vector for reading data from the buffer can be of any of the three types — vector, vectorf or vectorc.

rows=-1

[in] If the parameter is specified, the *cols* parameter should be specified as well. If the new matrix dimensions are not specified, the current ones are used. If the value is -1, then the number of rows does not change.

cols=-1

[in] If the parameter is not specified, the *rows* parameter should be skipped as well. The matrix adheres to the rule: either both parameters are specified, or none, otherwise an error will occur. If both parameters (*rows and cols*) are specified, the matrix size is changed. In case of -1, the number of columns does not change.

size=-1

[in] If the parameter is not specified or its value is -1, the vector length does not change.

Return Value

The number of read elements. 0 is returned in case of an error. For information about the error, use the GetLastError() function.

true if a matrix or a vector is handled successfully, otherwise false.

Note

For one-dimensional arrays, the number of the element, into which writing of data into an OpenCL buffer begins, is calculated taking into account the AS_SERIES flags.

An array of two or more dimensions is presented as one-dimensional. In this case, *data_offset* is the number of elements that should be skipped in the presentation, not the number of elements in the first dimension.

Example of calculating Pi using the equation:

$$\pi = \lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N-1}\frac{4}{1+\left(\frac{2k+1}{2N}\right)^2} = 16\lim_{N\to\infty}N\sum_{k=0}^{N-1}\frac{1}{4N^2+(2k+1)^2}$$

```
#define  _num_steps        1000000000
#define  _divisor          40000
#define  _step             1.0 / _num_steps
#define  _intrnCnt         _num_steps / _divisor
```

```
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
string D2S(double arg, int digits) { return DoubleToString(arg, digits); }
string I2S(int arg)                { return IntegerToString(arg); }

//--- OpenCL programm code
const string clSource=
  "#define _step "+D2S(_step, 12)+"                   \r\n"
  "#define _intrnCnt "+I2S(_intrnCnt)+"               \r\n"
  "                                                   \r\n"
  "__kernel void Pi( __global double *out )           \r\n"
  "{                                                  \r\n"
  "  int i = get_global_id( 0 );                      \r\n"
  "  double partsum = 0.0;                            \r\n"
  "  double x = 0.0;                                  \r\n"
  "  long from = i * _intrnCnt;                       \r\n"
  "  long to = from + _intrnCnt;                      \r\n"
  "  for( long j = from; j < to; j ++ )               \r\n"
  "  {                                                \r\n"
  "     x = ( j + 0.5 ) * _step;                      \r\n"
  "     partsum += 4.0 / ( 1. + x * x );              \r\n"
  "  }                                                \r\n"
  "  out[ i ] = partsum;                              \r\n"
  "}                                                  \r\n";

//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart()
 {
  Print("Pi Calculation: step = "+D2S(_step, 12)+"; _intrnCnt = "+I2S(_intrnCnt));
//--- prepare OpenCL contexts
  int clCtx;
  if((clCtx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
   {
    Print("OpenCL not found");
    return(-1);
   }
  int clPrg = CLProgramCreate(clCtx, clSource);
  int clKrn = CLKernelCreate(clPrg, "Pi");
  int clMem=CLBufferCreate(clCtx, _divisor*sizeof(double), CL_MEM_READ_WRITE);
  CLSetKernelArgMem(clKrn, 0, clMem);

  const uint offs[1]  = {0};
  const uint works[1] = {_divisor};
//--- launch OpenCL program
  ulong start=GetMicrosecondCount();
  if(!CLExecute(clKrn, 1, offs, works))
```

```
    {
     Print("CLExecute(clKrn, 1, offs, works) failed! Error ", GetLastError());
     CLFreeAll(clMem, clKrn, clPrg, clCtx);
     return(-1);
    }
//--- get results from OpenCL device
  vector buffer(_divisor);
  if(!CLBufferRead(clMem, 0, buffer))
    {
     Print("CLBufferRead(clMem, 0, buffer) failed! Error ", GetLastError());
     CLFreeAll(clMem, clKrn, clPrg, clCtx);
     return(-1);
    }
//--- sum all values to calculate Pi
  double Pi=buffer.Sum()*_step;

  double time=(GetMicrosecondCount()-start)/1000.;
  Print("OpenCL: Pi calculated for "+D2S(time, 2)+" ms");
  Print("Pi = "+DoubleToString(Pi, 12));
//--- free memory
  CLFreeAll(clMem, clKrn, clPrg, clCtx);
//--- success
  return(0);
 }
  /*
  Pi Calculation: step = 0.000000001000; _intrnCnt = 25000
  OpenCL: GPU device 'Ellesmere' selected
  OpenCL: Pi calculated for 99.98 ms
  Pi = 3.141592653590
  */
//+------------------------------------------------------------------+
//| Auxiliary routine to free memory                                 |
//+------------------------------------------------------------------+
void CLFreeAll(const int clMem, const int clKrn, const int clPrg, const int clCtx)
 {
  CLBufferFree(clMem);
  CLKernelFree(clKrn);
  CLProgramFree(clPrg);
  CLContextFree(clCtx);
 }
```

CLExecute

The function runs an OpenCL program. There are 3 versions of the function:

1. Launching kernel functions using one kernel

```
bool  CLExecute(
   int          kernel                    // Handle to the kernel of an OpenCL progra
   );
```

2. Launching several kernel copies (OpenCL function) with task space description

```
bool  CLExecute(
   int          kernel,                   // Handle to the kernel of an OpenCL progra
   uint         work_dim,                 // Dimension of the tasks space
   const uint&  global_work_offset[],     // Initial offset in the tasks space
   const uint&  global_work_size[]        // Total number of tasks
   );
```

3. Launching several kernel copies (OpenCL function) with task space description and specification of the size of the group's local task subset

```
bool  CLExecute(
   int          kernel,                   // Handle to the kernel of an OpenCL progra
   uint         work_dim,                 // Dimension of the tasks space
   const uint&  global_work_offset[],     // Initial offset in the tasks space
   const uint&  global_work_size[],       // Total number of tasks
   const uint&  local_work_size[]         // Number of tasks in the local group
   );
```

Parameters

kernel

[in] Handle to the OpenCL kernel.

work_dim

[in] Dimension of the tasks space.

global_work_offset[]

[in] Initial offset in the tasks space.

global_work_size[]

[in] The size of a subset of tasks.

local_work_size[]

[in] The size of the group's local task subset.

Return Value

Returns true if successful, otherwise returns false. For information about the error, use the GetLastError() function.

Note

Consider the use of the parameters in the following example:

- *work_dim* specifies the size of *work_items[]* array that describes the tasks. If *work_dim*=3, three-dimensional array *work_items[N1, N2, N3]* is used.
- *global_work_size[]* contains the values that set the *work_items[]* array size. If work_dim=3, *global_work_size[3]* array can be {40, 100, 320}. Then we have *work_items[40, 100, 320]*. So, the total number of tasks is 40 x 100 x 320 = 1 280 000.
- *local_work_size[]* sets the subset of the tasks that will be executed by the specified kernel of OpenCL program. Its size is equal to *work_items[]* size and allows to split the common task subset into smaller subsets without loss of remainder in division. In fact, the size of *local_work_size[]* array should be selected so that the *work_items[]* global task set will be split into smaller subsets. In this example *local_work_size[3]={10, 10, 10}* will be OK, as *work_items[40, 100, 320]* can be gathered from *local_items[10, 10, 10]* array without division remainder.

CLExecutionStatus

Returns the OpenCL program execution status.

```
int  CLExecutionStatus(
   int    kernel            // handle to a kernel of an OpenCL program
   );
```

Parameters

kernel

[in] Handle to a kernel of the OpenCL program.

Return Value

Returns the OpenCL program status. The value can be one of the following:

- CL_COMPLETE=0 – program complete,
- CL_RUNNING=1 – running,
- CL_SUBMITTED=2 – submitted for execution,
- CL_QUEUED=3 – queued,
- -1 (minus one) – error occurred when executing CLExecutionStatus().

Working with databases

The functions for working with databases apply the popular and easy-to-use SQLite engine. The convenient feature of this engine is that the entire database is located in a single file on a user PC's hard disk.

The functions allow for convenient creation of tables, adding data to them, performing modifications and sampling using simple SQL requests:

- receiving trading history and quotes from any formats,
- saving optimization and test results,
- preparing and exchanging data with other analysis packages,
- storing MQL5 application settings and status.

Queries allow using statistical and mathematical functions.

The functions for working with databases allow you to replace the most repetitive large data array handling operations with SQL requests, so that it is often possible to use the DatabaseExecute/DatabasePrepare calls instead of programming complex loops and comparisons. Use the DatabaseReadBind function to conveniently obtain query results in a ready-made structure. The function allows reading all record fields at once within a single call.

To accelerate reading, writing and modification, a database can be opened/created in RAM with the DATABASE_OPEN_MEMORY flag, although such a database is available only to a specific application and is not shared. When working with databases located on the hard disk, bulk data inserts/changes should be wrapped in transactions using DatabaseTransactionBegin/DatabaseTransactionCommit/DatabaseTransactionRollback. This accelerates the process hundreds of times.

To start working with the functions, read the article SQLite: Native handling of SQL databases in MQL5.

Function	Action
DatabaseOpen	Opens or creates a database in a specified file
DatabaseClose	Closes a database
DatabaseImport	Imports data from a file into a table
DatabaseExport	Exports a table or an SQL request execution result to a CSV file
DatabasePrint	Prints a table or an SQL request execution result in the Experts journal
DatabaseTableExists	Checks the presence of the table in a database
DatabaseExecute	Executes a request to a specified database
DatabasePrepare	Creates a handle of a request, which can then be executed using DatabaseRead()
DatabaseReset	Resets a request, like after calling DatabasePrepare()
DatabaseBind	Sets a parameter value in a request

Function	Action
DatabaseBindArray	Sets an array as a parameter value
DatabaseRead	Moves to the next entry as a result of a request
DatabaseReadBind	Moves to the next record and reads data into the structure from it
DatabaseFinalize	Removes a request created in DatabasePrepare()
DatabaseTransactionBegin	Starts transaction execution
DatabaseTransactionCommit	Completes transaction execution
DatabaseTransactionRollback	Rolls back transactions
DatabaseColumnsCount	Gets the number of fields in a request
DatabaseColumnName	Gets a field name by index
DatabaseColumnType	Gets a field type by index
DatabaseColumnSize	Gets a field size in bytes
DatabaseColumnText	Gets a field value as a string from the current record
DatabaseColumnInteger	Gets the int type value from the current record
DatabaseColumnLong	Gets the long type value from the current record
DatabaseColumnDouble	Gets the double type value from the current record
DatabaseColumnBlob	Gets a field value as an array from the current record

Statistical functions:

- mode - mode
- median - median (50th percentile)
- percentile_25 - 25th percentile
- percentile_75
- percentile_90
- percentile_95
- percentile_99
- stddev or stddev_samp — sample standard deviation
- stddev_pop — population standard deviation
- variance or var_samp — sample variance
- var_pop — population variance

Mathematical functions

- acos(X) - arccosine in radians
- acosh(X) - hyperbolic arccosine
- asin(X) - arcsine in radians

- asinh(X) - hyperbolic arcsine
- atan(X) - arctangent in radians
- atan2(X,Y) - arctangent in radians of the X/Y ratio
- atanh(X) - hyperbolic arctangent
- ceil(X) - rounding up to an integer
- ceiling(X) - rounding up to an integer
- cos(X) - angle cosine in radians
- cosh(X) - hyperbolic cosine
- degrees(X) - convert radians into the angle
- exp(X) - exponent
- floor(X) - rounding down to an integer
- ln(X) - natural logarithm
- log(B,X) - logarithm to the indicated base
- log(X) - decimal logarithm
- log10(X) - decimal logarithm
- log2(X) - logarithm to base 2
- mod(X,Y) - remainder of division
- pi() - approximate Pi
- pow(X,Y) - power by the indicated base
- power(X,Y) - power by the indicated base
- radians(X) - convert the angle into radians
- sin(X) - angle sine in radians
- sinh(X) - hyperbolic sine
- sqrt(X) - square root
- tan(X) - angle tangent in radians
- tanh(X) - hyperbolic tangent
- trunc(X) - truncate to an integer closest to 0

Example:

```
select
  count(*) as book_count,
  cast(avg(parent) as integer) as mean,
  cast(median(parent) as integer) as median,
  mode(parent) as mode,
  percentile_90(parent) as p90,
  percentile_95(parent) as p95,
  percentile_99(parent) as p99
from moz_bookmarks;
```

DatabaseOpen

Opens or creates a database in a specified file.

```
int  DatabaseOpen(
   string  filename,       // file name
   uint    flags           // combination of flags
   );
```

Parameters

filename

[in] File name relative to the "MQL5\Files" folder.

flags

[in] Combination of flags from the ENUM_DATABASE_OPEN_FLAGS enumeration.

Return Value

If executed successfully, the function returns the database handle, which is then used to access the database. Otherwise, it returns INVALID_HANDLE. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) - critical runtime error;
- ERR_WRONG_INTERNAL_PARAMETER (4002) - internal error, while accessing the "MQL5\Files" folder;
- ERR_INVALID_PARAMETER (4003) - path to the database file contains an empty string, or an incompatible combination of flags is set;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_WRONG_FILENAME (5002) - wrong database file name;
- ERR_TOO_LONG_FILENAME (5003) - absolute path to the database file exceeds the maximum length;
- ERR_DATABASE_TOO_MANY_OBJECTS (5122) - exceeded the maximum acceptable number of Database objects;
- ERR_DATABASE_CONNECT (5123) - database connection error;
- ERR_DATABASE_MISUSE (5621) - incorrect use of the SQLite library.

Note

If the *filename* parameter features NULL or the empty string "", a temporary file is created on the disk. It is automatically deleted after closing the database connection.

If the *filename* parameter features ":memory:", the database is created in the memory and is automatically deleted after the connection to it is closed.

If the *flags* parameter features none of the DATABASE_OPEN_READONLY or DATABASE_OPEN_READWRITE flags, the DATABASE_OPEN_READWRITE flag is used.

If the file extension is not specified, ".sqlite" is used.

ENUM_DATABASE_OPEN_FLAGS

ID	Description
DATABASE_OPEN_READONLY	Read only
DATABASE_OPEN_READWRITE	Open for reading and writing
DATABASE_OPEN_CREATE	Create the file on a disk if necessary
DATABASE_OPEN_MEMORY	Create a database in RAM
DATABASE_OPEN_COMMON	The file is in the common folder of all terminals

See also

DatabaseClose

DatabaseClose

Closes a database.

```
void  DatabaseClose(
   int  database      // database handle received in DatabaseOpen
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

Return Value

None.

Note

After calling DatabaseClose, all handles of requests to the database are automatically removed and become invalid.

If the handle is invalid, the function sets the ERR_DATABASE_INVALID_HANDLE error. You can check the error using GetLastError().

See also

DatabaseOpen, DatabasePrepare

DatabaseImport

Imports data from a file into a table.

```
long  DatabaseImport(
   int            database,          // database handle received in DatabaseOpen
   const string   table,             // name of a table to insert data
   const string   filename,          // name of a file to import data
   uint           flags,             // combination of flags
   const string   separator,         // data separator
   ulong          skip_rows,         // how many initial strings to skip
   const string   skip_comments      // string of characters defining comments
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

table

[in] Name of a table the data from a file is to be added to.

filename

[in] CSV file or ZIP archive for reading data. The name may contain subdirectories and is set relative to the MQL5\Files folder.

flags

[in] Combination of flags from the ENUM_DATABASE_IMPORT_FLAGS enumeration.

separator

[in] Data separator in CSV file.

skip_rows

[in] Number of initial strings to be skipped when reading data from the file.

skip_comments

[in] String of characters for designating strings as comments. If any character from *skip_comments* is detected at the beginning of a string, such a string is considered a comment and is not imported.

Return Value

Return the number of imported strings or -1 in case of an error. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) - no table name specified (empty string or NULL);
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle.

Note

If there is no table named *table*, it is generated automatically. Names and field types in the created table are defined automatically based on the file data.

If there is no table named *table*, it is generated automatically. Names and field types in the created table are defined automatically based on the file data.

ENUM_DATABASE_IMPORT_FLAGS

ID	Description
DATABASE_IMPORT_HEADER	The first line contains the names of the table fields
DATABASE_IMPORT_CRLF	CRLF (the default is LF) is considered a string break
DATABASE_IMPORT_APPEND	Add data to the end of an existing table
DATABASE_IMPORT_QUOTED_STRINGS	String values enclosed in double quotes
DATABASE_IMPORT_COMMON_FOLDER	The file is stored in the common folder of all client terminals \Terminal\Common\File.

Example of reading the table from the file created by the code from the DatabaseExport example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
 {
  string csv_filename;
//--- get the names of text files for downloading from the common folder of the client
  string filenames[];
  if(FileSelectDialog("Select a CSV file to download a table", NULL,
                      "Text files (*.csv)|*.csv",
                      FSD_WRITE_FILE|FSD_COMMON_FOLDER, filenames, "data.csv")>0)
   {
    //--- display the name of each selected file
    if(ArraySize(filenames)==1)
      csv_filename=filenames[0];
    else
     {
      Print("Unknown error while selecting file. Error code ", GetLastError());
      return;
     }
   }
  else
   {
    Print("CSV file not selected");
    return;
   }
//--- create or open a database
  string db_filename="test.sqlite";
  int db=DatabaseOpen(db_filename, DATABASE_OPEN_READWRITE|DATABASE_OPEN_CREATE);
//--- check if the TEST table exists
```

```
  if(DatabaseTableExists(db, "TEST"))
   {
    //--- remove the TEST table
    if(!DatabaseExecute(db, "DROP TABLE IF EXISTS TEST"))
     {
      Print("Failed to drop the TEST table with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   }
  //--- import entries from the file to the TEST table
  long imported=DatabaseImport(db, "TEST", csv_filename, DATABASE_IMPORT_HEADER|DATABA
  if(imported>0)
   {
    Print(imported," lines imported in table TEST");
    DatabasePrint(db,"SELECT * FROM TEST",DATABASE_PRINT_NO_INDEX);
   }
  else
    {
     Print("DatabaseImport() failed. Error ",GetLastError());
    }
//--- close the database file and inform of that
  DatabaseClose(db);
  PrintFormat("Database: %s closed", db_filename);
 }
```

See also

DatabaseOpen, DatabasePrint

DatabaseExport

Exports a table or an SQL request execution result to a CSV file. The file is created in the UTF-8 encoding.

```
long  DatabaseExport(
   int            database,           // database handle received in DatabaseOpen
   const string   table_or_sql,       // a table name or an SQL request
   const string   filename,           // a name of a CSV file for data export
   uint           flags,              // combination of flags
   const string   separator           // data separator in the CSV file
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

table_or_sql

[in] A name of a table or a text of an SQL request whose results are to be exported to a specified file.

filename

[in] A file name for data export. The path is set relative to the MQL5\Files folder.

flags

[in] Combination of flags from the ENUM_DATABASE_EXPORT_FLAGS enumeration.

separator

[in] Data separator. If NULL is specified, the '\t' tabulation character is used as a separator. An empty string "" is considered a valid separator but the obtained CSV file cannot be read as a table – it is considered as a set of strings.

Return Value

Return the number of exported entries or a negative value in case of an error. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) – critical runtime error;
- ERR_INVALID_PARAMETER (4003) – path to the database file contains an empty string, or an incompatible combination of flags is set;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_FUNCTION_NOT_ALLOWED(4014) – specified pipe is not allowed;
- ERR_PROGRAM_STOPPED(4022) – operation canceled (MQL program stopped);
- ERR_WRONG_FILENAME (5002) - invalid file name;
- ERR_TOO_LONG_FILENAME (5003) - absolute path to the file exceeds the maximum length;
- ERR_CANNOT_OPEN_FILE(5004) – unable to open the file for writing;
- ERR_FILE_WRITEERROR(5026) – unable to write to the file;
- ERR_DATABASE_INTERNAL (5120) – internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;

- ERR_DATABASE_QUERY_PREPARE(5125) - request generation error;
- ERR_DATABASE_QUERY_NOT_READONLY - read-only request is allowed.

Note

If request results are exported, the SQL request should begin with "SELECT" or "select". In other words, the SQL request cannot alter the database status, otherwise DatabaseExport() fails with an error.

Database string values may contain the conversion character ('\r' or '\r\n'), as well as the value separator character set in the *separator* parameter. In this case, be sure to use the DATABASE_EXPORT_QUOTED_STRINGS flag in the 'flags' parameter. If this flag is present, all displayed strings are enclosed in double quotes. If a string contains a double quote, it is replaced by two double quotes.

ENUM_DATABASE_EXPORT_FLAGS

ID	Description
DATABASE_EXPORT_HEADER	Display field names in the first string
DATABASE_EXPORT_INDEX	Display string indices
DATABASE_EXPORT_NO_BOM	Do not insert BOM mark at the beginning of the file (BOM is inserted by default)
DATABASE_EXPORT_CRLF	Use CRLF for string break (the default is LF)
DATABASE_EXPORT_APPEND	Add data to the end of an existing file (by default, the file is overwritten). If the file does not exist, it will be created.
DATABASE_EXPORT_QUOTED_STRINGS	Display string values in double quotes.
DATABASE_EXPORT_COMMON_FOLDER	A CSV file is created in the common folder of all client terminals \Terminal\Common\File.

Example:

```
input int InpRates=100;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlRates rates[];
//--- remember the start time before receiving bars
   ulong start=GetMicrosecondCount();
//--- request the last 100 bars on H1
   if(CopyRates(Symbol(), PERIOD_H1, 1, InpRates, rates)<InpRates)
     {
      Print("CopyRates() failed,, Error ", GetLastError());
```

```
    return;
   }
  else
   {
    //--- how many bars were received and how much time it took to receive them
    PrintFormat("%s: CopyRates received %d bars in %d ms ",
                _Symbol, ArraySize(rates), (GetMicrosecondCount()-start)/1000);
   }
//--- set the file name for storing the database
  string filename=_Symbol+"_"+EnumToString(PERIOD_H1)+"_"+TimeToString(TimeCurrent())
  StringReplace(filename, ":", "-"); // ":" character is not allowed in file names
//--- open/create the database in the common terminal folder
  int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE|DATABASE_OPEN_CREATE|DATABASE_
  if(db==INVALID_HANDLE)
   {
    Print("Database: ", filename, " open failed with code ", GetLastError());
    return;
   }
  else
    Print("Database: ", filename, " opened successfully");

//--- check if the RATES table exists
  if(DatabaseTableExists(db, "RATES"))
   {
    //--- remove the RATES table
    if(!DatabaseExecute(db, "DROP TABLE IF EXISTS RATES"))
     {
      Print("Failed to drop the RATES table with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   }
//--- create the RATES table
  if(!DatabaseExecute(db, "CREATE TABLE RATES("
                      "SYMBOL             CHAR(10),"
                      "TIME               INT NOT NULL,"
                      "OPEN               REAL,"
                      "HIGH               REAL,"
                      "LOW                REAL,"
                      "CLOSE              REAL,"
                      "TICK_VOLUME        INT,"
                      "SPREAD             INT,"
                      "REAL_VOLUME        INT);"))
   {
    Print("DB: ", filename, " create table RATES with code ", GetLastError());
    DatabaseClose(db);
    return;
   }
//--- display the list of all fields in the RATES table
```

```
  if(DatabasePrint(db, "PRAGMA TABLE_INFO(RATES)", 0)<0)
   {
    PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(RATES)\") failed, error code=%d at
    DatabaseClose(db);
    return;
   }
//--- create a parametrized request to add bars to the RATES table
  string sql="INSERT INTO RATES (SYMBOL,TIME,OPEN,HIGH,LOW,CLOSE,TICK_VOLUME,SPREAD,R
             " VALUES (?1,?2,?3,?4,?5,?6,?7,?8,?9)"; // request parameters
  int request=DatabasePrepare(db, sql);
  if(request==INVALID_HANDLE)
   {
    PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
    Print("SQL request: ", sql);
    DatabaseClose(db);
    return;
   }
//--- set the value of the first request parameter
  DatabaseBind(request, 0, _Symbol);
//--- remember the start time before adding bars to the RATES table
  start=GetMicrosecondCount();
  DatabaseTransactionBegin(db);
  int total=ArraySize(rates);
  bool request_error=false;
  for(int i=0; i<total; i++)
   {
    //--- set the values of the remaining parameters before adding the entry
    ResetLastError();
    if(!DatabaseBind(request, 1, rates[i].time))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    //--- if the previous DatabaseBind() call was successful, set the next parameter
    if(!request_error && !DatabaseBind(request, 2, rates[i].open))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
     }
    if(!request_error && !DatabaseBind(request, 3, rates[i].high))
     {
      PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
      PrintFormat("Bar #%d line=%d", i+1, __LINE__);
      request_error=true;
      break;
```

```
       }
      if(!request_error && !DatabaseBind(request, 4, rates[i].low))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
      if(!request_error && !DatabaseBind(request, 5, rates[i].close))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
      if(!request_error && !DatabaseBind(request, 6, rates[i].tick_volume))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
      if(!request_error && !DatabaseBind(request, 7, rates[i].spread))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }
      if(!request_error && !DatabaseBind(request, 8, rates[i].real_volume))
       {
        PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
        PrintFormat("Bar #%d line=%d", i+1, __LINE__);
        request_error=true;
        break;
       }

      //--- execute a request for inserting the entry and check for an error
      if(!request_error && !DatabaseRead(request) && (GetLastError()!=ERR_DATABASE_NO_M
       {
        PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
        DatabaseFinalize(request);
        request_error=true;
        break;
       }
      //--- reset the request before the next parameter update
      if(!request_error && !DatabaseReset(request))
       {
        PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
      DatabaseFinalize(request);
      request_error=true;
      break;
     }
   } //--- done going through all the bars

//--- transactions status
  if(request_error)
   {
    PrintFormat("Table RATES: failed to add %d bars ", ArraySize(rates));
    DatabaseTransactionRollback(db);
    DatabaseClose(db);
    return;
   }
  else
   {
    DatabaseTransactionCommit(db);
    PrintFormat("Table RATES: added %d bars in %d ms",
                ArraySize(rates), (GetMicrosecondCount()-start)/1000);
   }
//--- save the RATES table to a CSV file
  string csv_filename=Symbol()+".csv";
  long saved=DatabaseExport(db, "SELECT * FROM RATES", csv_filename, DATABASE_EXPORT_
  if(saved>0)
    Print("Table RATES saved in ", Symbol(), ".csv");
  else
    Print("DatabaseExport() failed. Error ", GetLastError());
//--- close the database file and inform of that
  DatabaseClose(db);
  PrintFormat("Database: %s created and closed", filename);
```

See also

DatabasePrint, DatabaseImport

DatabasePrint

Prints a table or an SQL request execution result in the Experts journal.

```
long  DatabasePrint(
   int            database,          // database handle received in DatabaseOpen
   const string   table_or_sql,      // a table or an SQL request
   uint           flags              // combination of flags
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

table_or_sql

[in] A name of a table or a text of an SQL request whose results are displayed in the Experts journal.

flags

[in] Combination of flags defining the output formatting. The flags are defined as follows:

DATABASE_PRINT_NO_HEADER - do not display table column names (field names)
DATABASE_PRINT_NO_INDEX - do not display string indices
DATABASE_PRINT_NO_FRAME - do not display a frame separating a header and data
DATABASE_PRINT_STRINGS_RIGHT - align strings to the right.

If flags=0, the columns and the strings are displayed, the header and the data are separated by the frame, while the strings are aligned to the left.

Return Value

Return the number of exported strings or -1 in case of an error. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) - critical runtime error;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;

Note

If the journal displays request results, the SQL request should begin with "SELECT" or "select". In other words, the SQL request cannot alter the database status, otherwise DatabasePrint() fails with an error.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string filename="departments.sqlite";
```

```
//--- create or open the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE |DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }

//--- create the COMPANY table
   if(!CreataTableCompany(db))
     {
      DatabaseClose(db);
      return;
     }
//--- create the DEPARTMENT table
   if(!CreataTableDepartment(db))
     {
      DatabaseClose(db);
      return;
     }

//--- display the list of all fields in the COMPANY and DEPARTMENT tables
   PrintFormat("Try to print request \"PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(D
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(DEPARTMENT)", 0
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO()\") failed, error code=%d", Get
      DatabaseClose(db);
      return;
     }
//--- display the COMPANY table in the log
   PrintFormat("Try to print request \"SELECT * from COMPANY\"");
   if(DatabasePrint(db, "SELECT * from COMPANY", 0)<0)
     {
      Print("DatabasePrint failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- request text for combining the COMPANY and DEPARTMENT tables
   string request="SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT 
                  "ON COMPANY.ID = DEPARTMENT.EMP_ID";
//--- display the table combining result
   PrintFormat("Try to print request \"SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OU
   if(DatabasePrint(db, request, 0)<0)
     {
      Print("DatabasePrint failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- close the database
```

```
   DatabaseClose(db);
  }
/*
Conclusion:
Try to print request "PRAGMA TABLE_INFO(COMPANY);PRAGMA TABLE_INFO(DEPARTMENT)"
#| cid name    type     notnull dflt_value pk
-+-------------------------------------------
1|   0 ID      INT            1             1
2|   1 NAME    TEXT           1             0
3|   2 AGE     INT            1             0
4|   3 ADDRESS CHAR(50)       0             0
5|   4 SALARY  REAL           0             0
#| cid name   type     notnull dflt_value pk
-+------------------------------------------
1|   0 ID     INT            1             1
2|   1 DEPT   CHAR(50)       1             0
3|   2 EMP_ID INT            1             0
Try to print request "SELECT * from COMPANY"
#| ID NAME  AGE ADDRESS     SALARY
-+--------------------------------
1|  1 Paul   32 California 25000.0
2|  2 Allen  25 Texas      15000.0
3|  3 Teddy  23 Norway     20000.0
4|  4 Mark   25 Rich-Mond  65000.0
5|  5 David  27 Texas      85000.0
6|  6 Kim    22 South-Hall 45000.0
7|  7 James  24 Houston    10000.0
Try to print request "SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMEN
#| EMP_ID NAME  DEPT
-+-------------------------
1|      1 Paul  IT Billing
2|      2 Allen Engineering
3|        Teddy
4|        Mark
5|        David
6|        Kim
7|      7 James Finance
*/
//+------------------------------------------------------------------+
//| Create the COMPANY table                           |
//+------------------------------------------------------------------+
bool CreateTableCompany(int database)
  {
//--- if the COMPANY table exists, delete it
   if(DatabaseTableExists(database, "COMPANY"))
     {
      //--- delete the table
      if(!DatabaseExecute(database, "DROP TABLE COMPANY"))
        {
         Print("Failed to drop table COMPANY with code ", GetLastError());
         return(false);
        }
     }
//--- create the COMPANY table
   if(!DatabaseExecute(database, "CREATE TABLE COMPANY("
                       "ID INT PRIMARY KEY     NOT NULL,"
                       "NAME           TEXT    NOT NULL,"
                       "AGE            INT     NOT NULL,"
                       "ADDRESS        CHAR(50),"
                       "SALARY         REAL );"))
     {
      Print("DB: create table COMPANY failed with code ", GetLastError());
      return(false);
     }

//--- enter data to the COMPANY table
   if(!DatabaseExecute(database, "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VA
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (5, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (6, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (7, '
     {
      Print("COMPANY insert failed with code ", GetLastError());
      return(false);
     }
//--- success
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the DEPARTMENT table                                       |
//+------------------------------------------------------------------+
bool CreateTableDepartment(int database)
  {
//--- if the DEPARTMENT table exists, delete it
   if(DatabaseTableExists(database, "DEPARTMENT"))
     {
      //--- delete the table
      if(!DatabaseExecute(database, "DROP TABLE DEPARTMENT"))
        {
         Print("Failed to drop table DEPARTMENT  with code ", GetLastError());
         return(false);
        }
     }
//--- create the DEPARTMENT table
   if(!DatabaseExecute(database, "CREATE TABLE DEPARTMENT ("
                       "ID      INT PRIMARY KEY    NOT NULL,"
```

```
                        "DEPT     CHAR(50)          NOT NULL,"
                        "EMP_ID   INT               NOT NULL);"))
      {
       Print("DB: create table DEPARTMENT failed with code ", GetLastError());
       return(false);
      }

//--- enter data to the DEPARTMENT table
   if(!DatabaseExecute(database, "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (1, 
                        "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (2, 'Engineeri
                        "INSERT INTO DEPARTMENT (ID,DEPT,EMP_ID) VALUES (3, 'Finance', 
      {
       Print("DEPARTMENT insert failed with code ", GetLastError());
       return(false);
      }
//--- success
   return(true);
  }
//+------------------------------------------------------------------
```

See also

DatabaseExport, DatabaseImport

DatabaseTableExists

Checks the presence of the table in a database.

```
bool  DatabaseTableExists(
   int      database,      // database handle received in DatabaseOpen
   string   table          // table name
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

table

[in] Table name.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) – no table name specified (empty string or NULL);
- ERR_WRONG_STRING_PARAMETER (5040) – error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) – internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_EXECUTE (5124) - request execution error;
- ERR_DATABASE_NO_MORE_DATA (5126) - no table exists (not an error, normal completion).

See also

DatabasePrepare, DatabaseFinalize

DatabaseExecute

Executes a request to a specified database.

```
bool  DatabaseExecute(
   int      database,      // database handle received in DatabaseOpen
   string   sql            // SQL request
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

sql

[in] SQL request.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) - critical runtime error;
- ERR_INVALID_PARAMETER (4003) - sql parameter contains an empty string;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_WRONG_STRING_PARAMETER (5040) - error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_EXECUTE (5124) - request execution error.

Example:

```
//--- symbol statistics
struct Symbol_Stats
  {
   string            name;             // symbol name
   int               trades;           // number of trades for the symbol
   double            gross_profit;     // total profit for the symbol
   double            gross_loss;       // total loss for the symbol
   double            total_commission; // total commission for the symbol
   double            total_swap;       // total swaps for the symbol
   double            total_profit;     // total profit excluding swaps and commissions
   double            net_profit;       // net profit taking into account swaps and co
   int               win_trades;       // number of profitable trades
   int               loss_trades;      // number of losing trades
   double            expected_payoff;  // expected payoff for the trade excluding swap
   double            win_percent;      // percentage of winning trades
   double            loss_percent;     // percentage of losing trades
   double            average_profit;   // average profit
   double            average_loss;     // average loss
   double            profit_factor;    // profit factor
  };

//--- Magic Number statistics
struct Magic_Stats
  {
   long                 magic;            // EA's Magic Number
   int                  trades;           // number of trades for the symbol
   double               gross_profit;     // total profit for the symbol
   double               gross_loss;       // total loss for the symbol
   double               total_commission; // total commission for the symbol
   double               total_swap;       // total swaps for the symbol
   double               total_profit;     // total profit excluding swaps and commissions
   double               net_profit;       // net profit taking into account swaps and con
   int                  win_trades;       // number of profitable trades
   int                  loss_trades;      // number of losing trades
   double               expected_payoff;  // expected payoff for the trade excluding swa
   double               win_percent;      // percentage of winning trades
   double               loss_percent;     // percentage of losing trades
   double               average_profit;   // average profit
   double               average_loss;     // average loss
   double               profit_factor;    // profit factor
  };

//--- entry hour statistics
struct Hour_Stats
  {
   char                 hour_in;          // market entry hour
   int                  trades;           // number of trades in this entry hour
   double               volume;           // volume of trades in this entry hour
   double               gross_profit;     // total profit in this entry hour
   double               gross_loss;       // total loss in this entry hour
   double               net_profit;       // net profit taking into account swaps and con
   int                  win_trades;       // number of profitable trades
   int                  loss_trades;      // number of losing trades
   double               expected_payoff;  // expected payoff for the trade excluding swa
   double               win_percent;      // percentage of winning trades
   double               loss_percent;     // percentage of losing trades
   double               average_profit;   // average profit
   double               average_loss;     // average loss
   double               profit_factor;    // profit factor
  };

int ExtDealsTotal=0;;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
```

```
   string filename=IntegerToString(AccountInfoInteger(ACCOUNT_LOGIN))+"_stats.sqlite";
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- create the DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }
   PrintFormat("Deals in the trading history: %d ", ExtDealsTotal);

//--- get trading statistics per symbols
   int request=DatabasePrepare(db, "SELECT r.*,"
                               "   (case when r.trades != 0 then (r.gross_profit+r.gr
                               "   (case when r.trades != 0 then r.win_trades*100.0/r.
                               "   (case when r.trades != 0 then r.loss_trades*100.0/r
                               "   r.gross_profit/r.win_trades as average_profit,"
                               "   r.gross_loss/r.loss_trades as average_loss,"
                               "   (case when r.gross_loss!=0.0 then r.gross_profit/(
                               "FROM "
                               "   ("
                               "   SELECT SYMBOL,"
                               "   sum(case when entry =1 then 1 else 0 end) as trades
                               "   sum(case when profit > 0 then profit else 0 end) as
                               "   sum(case when profit < 0 then profit else 0 end) as
                               "   sum(swap) as total_swap,"
                               "   sum(commission) as total_commission,"
                               "   sum(profit) as total_profit,"
                               "   sum(profit+swap+commission) as net_profit,"
                               "   sum(case when profit > 0 then 1 else 0 end) as win_
                               "   sum(case when profit < 0 then 1 else 0 end) as loss
                               "   FROM DEALS "
                               "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                               "   GROUP BY SYMBOL"
                               "   ) as r");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Symbol_Stats stats[], symbol_stats;
   ArrayResize(stats, ExtDealsTotal);
   int i=0;
```

```
//--- get records from request results
   for(; DatabaseReadBind(request, symbol_stats) ; i++)
     {
      stats[i].name=symbol_stats.name;
      stats[i].trades=symbol_stats.trades;
      stats[i].gross_profit=symbol_stats.gross_profit;
      stats[i].gross_loss=symbol_stats.gross_loss;
      stats[i].total_commission=symbol_stats.total_commission;
      stats[i].total_swap=symbol_stats.total_swap;
      stats[i].total_profit=symbol_stats.total_profit;
      stats[i].net_profit=symbol_stats.net_profit;
      stats[i].win_trades=symbol_stats.win_trades;
      stats[i].loss_trades=symbol_stats.loss_trades;
      stats[i].expected_payoff=symbol_stats.expected_payoff;
      stats[i].win_percent=symbol_stats.win_percent;
      stats[i].loss_percent=symbol_stats.loss_percent;
      stats[i].average_profit=symbol_stats.average_profit;
      stats[i].average_loss=symbol_stats.average_loss;
      stats[i].profit_factor=symbol_stats.profit_factor;
     }
   ArrayResize(stats, i);
   Print("Trade statistics by Symbol");
   ArrayPrint(stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- get trading statistics for Expert Advisors by Magic Numbers
   request=DatabasePrepare(db, "SELECT r.*,"
                           "   (case when r.trades != 0 then (r.gross_profit+r.gross_
                           "   (case when r.trades != 0 then r.win_trades*100.0/r.tra
                           "   (case when r.trades != 0 then r.loss_trades*100.0/r.tr
                           "   r.gross_profit/r.win_trades as average_profit,"
                           "   r.gross_loss/r.loss_trades as average_loss,"
                           "   (case when r.gross_loss!=0.0 then r.gross_profit/(-r.g
                           "FROM "
                           "   ("
                           "   SELECT MAGIC,"
                           "   sum(case when entry =1 then 1 else 0 end) as trades,"
                           "   sum(case when profit > 0 then profit else 0 end) as gr
                           "   sum(case when profit < 0 then profit else 0 end) as gr
                           "   sum(swap) as total_swap,"
                           "   sum(commission) as total_commission,"
                           "   sum(profit) as total_profit,"
                           "   sum(profit+swap+commission) as net_profit,"
                           "   sum(case when profit > 0 then 1 else 0 end) as win_tra
                           "   sum(case when profit < 0 then 1 else 0 end) as loss_tr
                           "   FROM DEALS "
                           "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                           "   GROUP BY MAGIC"
                           "   ) as r");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Magic_Stats EA_stats[], magic_stats;
   ArrayResize(EA_stats, ExtDealsTotal);
   i=0;
//--- print
   for(; DatabaseReadBind(request, magic_stats) ; i++)
     {
      EA_stats[i].magic=magic_stats.magic;
      EA_stats[i].trades=magic_stats.trades;
      EA_stats[i].gross_profit=magic_stats.gross_profit;
      EA_stats[i].gross_loss=magic_stats.gross_loss;
      EA_stats[i].total_commission=magic_stats.total_commission;
      EA_stats[i].total_swap=magic_stats.total_swap;
      EA_stats[i].total_profit=magic_stats.total_profit;
      EA_stats[i].net_profit=magic_stats.net_profit;
      EA_stats[i].win_trades=magic_stats.win_trades;
      EA_stats[i].loss_trades=magic_stats.loss_trades;
      EA_stats[i].expected_payoff=magic_stats.expected_payoff;
      EA_stats[i].win_percent=magic_stats.win_percent;
      EA_stats[i].loss_percent=magic_stats.loss_percent;
      EA_stats[i].average_profit=magic_stats.average_profit;
      EA_stats[i].average_loss=magic_stats.average_loss;
      EA_stats[i].profit_factor=magic_stats.profit_factor;
     }
   ArrayResize(EA_stats, i);
   Print("Trade statistics by Magic Number");
   ArrayPrint(EA_stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- make sure that hedging system for open position management is used on the accou
   if((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)!=ACCOUNT_MARG
     {
      //--- deals cannot be transformed to trades using a simple method through transa
      DatabaseClose(db);
      return;
     }

//--- now create the TRADES table based on the DEALS table
   if(!CreateTableTrades(db))
     {
      DatabaseClose(db);
      return;
     }
//--- fill in the TRADES table using an SQL query based on DEALS table data
   if(DatabaseTableExists(db, "DEALS"))
      //--- populate the TRADES table
      if(!DatabaseExecute(db, "INSERT INTO TRADES(TIME_IN,HOUR_IN,TICKET,TYPE,VOLUME,S
                          "SELECT "
                          "   d1.time as time_in,"
                          "   d1.hour as hour_in,"
                          "   d1.position_id as ticket,"
                          "   d1.type as type,"
                          "   d1.volume as volume,"
                          "   d1.symbol as symbol,"
                          "   d1.price as price_in,"
                          "   d2.time as time_out,"
                          "   d2.price as price_out,"
                          "   d1.commission+d2.commission as commission,"
                          "   d2.swap as swap,"
                          "   d2.profit as profit "
                          "FROM DEALS d1 "
                          "INNER JOIN DEALS d2 ON d1.position_id=d2.position_id "
                          "WHERE d1.entry=0 AND d2.entry=1     "))
        {
         Print("DB: fillng the table TRADES failed with code ", GetLastError());
         return;
        }

//--- get trading statistics by market entry hours
   request=DatabasePrepare(db, "SELECT r.*,"
                           "    (case when r.trades != 0 then (r.gross_profit+r.gross_l
                           "    (case when r.trades != 0 then r.win_trades*100.0/r.trad
                           "    (case when r.trades != 0 then r.loss_trades*100.0/r.tra
                           "   r.gross_profit/r.win_trades as average_profit,"
                           "   r.gross_loss/r.loss_trades as average_loss,"
                           "    (case when r.gross_loss!=0.0 then r.gross_profit/(-r.gr
                           "FROM "
                           "    ("
                           "   SELECT HOUR_IN,"
                           "   count() as trades,"
                           "   sum(volume) as volume,"
                           "   sum(case when profit > 0 then profit else 0 end) as gro
                           "   sum(case when profit < 0 then profit else 0 end) as gro
                           "   sum(profit) as net_profit,"
                           "   sum(case when profit > 0 then 1 else 0 end) as win_trad
                           "   sum(case when profit < 0 then 1 else 0 end) as loss_tra
                           "   FROM TRADES "
                           "   WHERE SYMBOL <> '' and SYMBOL is not NULL "
                           "   GROUP BY HOUR_IN"
                           "   ) as r");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   Hour_Stats hours_stats[], h_stats;
   ArrayResize(hours_stats, ExtDealsTotal);
   i=0;
//--- print
   for(; DatabaseReadBind(request, h_stats) ; i++)
     {
      hours_stats[i].hour_in=h_stats.hour_in;
      hours_stats[i].trades=h_stats.trades;
      hours_stats[i].volume=h_stats.volume;
      hours_stats[i].gross_profit=h_stats.gross_profit;
      hours_stats[i].gross_loss=h_stats.gross_loss;
      hours_stats[i].net_profit=h_stats.net_profit;
      hours_stats[i].win_trades=h_stats.win_trades;
      hours_stats[i].loss_trades=h_stats.loss_trades;
      hours_stats[i].expected_payoff=h_stats.expected_payoff;
      hours_stats[i].win_percent=h_stats.win_percent;
      hours_stats[i].loss_percent=h_stats.loss_percent;
      hours_stats[i].average_profit=h_stats.average_profit;
      hours_stats[i].average_loss=h_stats.average_loss;
      hours_stats[i].profit_factor=h_stats.profit_factor;
     }
   ArrayResize(hours_stats, i);
   Print("Trade statistics by entry hour");
   ArrayPrint(hours_stats);
   Print("");
//--- delete the request
   DatabaseFinalize(request);

//--- close database
   DatabaseClose(db);
   return;
  }
/*
Deals in the trading history: 2771
Trade statistics by Symbol
      [name] [trades] [gross_profit] [gross_loss] [total_commission] [total_swap] [tot
[0] "AUDUSD"      112      503.20000   -568.00000           -8.83000    -24.64000
[1] "EURCHF"      125      607.71000   -956.85000          -11.77000    -45.02000
[2] "EURJPY"      127     1078.49000  -1057.83000          -10.61000    -45.76000
[3] "EURUSD"      233     1685.60000  -1386.80000          -41.00000    -83.76000
[4] "GBPCHF"      125     1881.37000  -1424.72000          -22.60000    -51.56000
[5] "GBPJPY"      127     1943.43000  -1776.67000          -18.84000    -52.46000
```

```
[6] "GBPUSD"      121     1668.50000  -1438.20000           -7.96000     -49.93000
[7] "USDCAD"       99      405.28000   -475.47000           -8.68000     -31.68000
[8] "USDCHF"      206     1588.32000  -1241.83000          -17.98000     -65.92000
[9] "USDJPY"      107      464.73000   -730.64000          -35.12000     -34.24000

Trade statistics by Magic Number
    [magic] [trades] [gross_profit] [gross_loss] [total_commission] [total_swap] [tota
[0]     100      242     2584.80000  -2110.00000          -33.36000    -93.53000
[1]     200      254     3021.92000  -2834.50000          -29.45000    -98.22000
[2]     300      250     2489.08000  -2381.57000          -34.37000    -96.58000
[3]     400      224     1272.50000  -1283.00000          -24.43000    -64.80000
[4]     500      198     1141.23000  -1051.91000          -27.66000    -63.36000
[5]     600      214     1317.10000  -1396.03000          -34.12000    -68.48000

Trade statistics by entry hour
     [hour_in] [trades] [volume] [gross_profit] [gross_loss] [net_profit] [win_trades]
[ 0]         0       50  5.00000      336.51000   -747.47000   -410.96000            2
[ 1]         1       20  2.00000      102.56000    -57.20000     45.36000            1
[ 2]         2        6  0.60000       38.55000    -14.60000     23.95000
[ 3]         3       38  3.80000      173.84000   -200.15000    -26.31000            2
[ 4]         4       60  6.00000      361.44000   -389.40000    -27.96000            2
[ 5]         5       32  3.20000      157.43000   -179.89000    -22.46000            2
[ 6]         6       18  1.80000       95.59000   -162.33000    -66.74000            1
[ 7]         7       14  1.40000       38.48000   -134.30000    -95.82000
[ 8]         8       42  4.20000      368.48000   -322.30000     46.18000            2
[ 9]         9      118 11.80000     1121.62000   -875.21000    246.41000            7
[10]        10      206 20.60000     2280.59000  -2021.80000    258.79000           11
[11]        11      138 13.80000     1377.02000   -994.18000    382.84000            8
[12]        12      152 15.20000     1247.56000  -1463.80000   -216.24000            8
[13]        13       64  6.40000      778.27000   -516.22000    262.05000            3
[14]        14       62  6.20000      536.93000   -427.47000    109.46000            3
[15]        15       50  5.00000      699.92000   -413.00000    286.92000            2
[16]        16       88  8.80000      778.55000   -514.00000    264.55000            5
[17]        17       76  7.60000      533.92000  -1019.46000   -485.54000            4
[18]        18       52  5.20000      237.17000   -246.78000     -9.61000            2
[19]        19       52  5.20000      407.67000   -150.36000    257.31000            3
[20]        20       18  1.80000       65.92000    -89.09000    -23.17000
[21]        21       10  1.00000       41.86000    -32.38000      9.48000
[22]        22       14  1.40000       45.55000    -83.72000    -38.17000
[23]        23        2  0.20000        1.20000     -1.90000     -0.70000
*/

//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(database, "DEALS"))
```

```
     {
      return(false);
     }
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID          INT KEY NOT NULL,"
                          "ORDER_ID    INT     NOT NULL,"
                          "POSITION_ID INT     NOT NULL,"
                          "TIME        INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "ENTRY       INT     NOT NULL,"
                          "SYMBOL      CHAR(10),"
                          "VOLUME      REAL,"
                          "PRICE       REAL,"
                          "PROFIT      REAL,"
                          "SWAP        REAL,"
                          "COMMISSION  REAL,"
                          "MAGIC       INT,"
                          "HOUR        INT,"
                          "REASON      INT);"))
        {
         Print("DB: create the DEALS table  failed with code ", GetLastError());
         return(false);
        }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   ExtDealsTotal=HistoryDealsTotal();
//--- add deals to the table
   if(!InsertDeals(database))
      return(false);
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database        |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
     {
      Print("Failed to drop the DEALS table with code ", GetLastError());
      return(false);
     }
//--- the table has been successfully deleted
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database)
  {
//--- Auxiliary variables
   ulong    deal_ticket;         // deal ticket
   long     order_ticket;        // the ticket of the order by which the deal was exe
   long     position_ticket;     // ID of the position to which the deal belongs
   datetime time;                // deal execution time
   long     type ;               // deal type
   long     entry ;              // deal direction
   string   symbol;              // the symbol fro which the deal was executed
   double   volume;              // operation volume
   double   price;               // price
   double   profit;              // financial result
   double   swap;                // swap
   double   commission;          // commission
   long     magic;               // Magic number (Expert Advisor ID)
   long     reason;              // deal execution reason or source
   char     hour;                // deal execution hour
   MqlDateTime time_strusture;
//--- go through all deals and add them to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
// --- lock the database before executing transactions
   DatabaseTransactionBegin(database);
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      TimeToStruct(time, time_strusture);
      hour= (char)time_strusture.hour;
      //--- add each deal to the table using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TI
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G, 
                                       deal_ticket, order_ticket, position_ticket, tir
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%d with code %d", __FUNCTION__, deal_
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
         break;
        }
     }
//--- check for transaction execution errors
   if(failed)
     {
      //--- roll back all transactions and unlock the database
      DatabaseTransactionRollback(database);
      PrintFormat("%s: DatabaseExecute() failed with code ", __FUNCTION__, GetLastErro
      return(false);
     }
//--- all transactions have been performed successfully - record changes and unlock th
   DatabaseTransactionCommit(database);
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the TRADES table                                         |
//+------------------------------------------------------------------+
bool CreateTableTrades(int database)
  {
//--- if the TRADES table already exists, delete it
   if(!DeleteTable(database, "TRADES"))
      return(false);
//--- check if the table exists
   if(!DatabaseTableExists(database, "TRADES"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE TRADES("
                          "TIME_IN     INT     NOT NULL,"
                          "HOUR_IN     INT     NOT NULL,"
                          "TICKET      INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "VOLUME      REAL,"
                          "SYMBOL      CHAR(10),"
                          "PRICE_IN    REAL,"
                          "TIME_OUT    INT     NOT NULL,"
                          "PRICE_OUT   REAL,"
                          "COMMISSION  REAL,"
                          "SWAP        REAL,"
                          "PROFIT      REAL);"))
        {
         Print("DB: create the TRADES table failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
```

See also

[DatabasePrepare](), [DatabaseFinalize]()

DatabasePrepare

Creates a handle of a request, which can then be executed using DatabaseRead().

```
int  DatabasePrepare(
   int      database,       // database handle received in DatabaseOpen
   string   sql,            // SQL request
            ...             // request parameters
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

sql

[in] SQL request that may contain automatically substituted parameters named ?1,?2,...

...

[in] Automatically substituted request parameters.

Return Value

If successful, the function returns a handle for the SQL request. Otherwise, it returns INVALID_HANDLE. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) - path to the database file contains an empty string, or an incompatible combination of flags is set;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_WRONG_STRING_PARAMETER (5040) - error converting a request into a UTF-8 string;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_TOO_MANY_OBJECTS (5122) - exceeded the maximum acceptable number of Database objects;
- ERR_DATABASE_PREPARE (5125) - request generation error.

Note

The DatabasePrepare() function does not perform a request to a database. Its purpose is to verify the request parameters and return the handle for executing the SQL request based on the verification results. The request itself is set during the DatabaseRead() first call.

Example:

```
//--- Structure to store the deal
struct Deal
  {
   ulong             ticket;           // DEAL_TICKET
   long              order_ticket;     // DEAL_ORDER
   long              position_ticket;  // DEAL_POSITION_ID
   datetime          time;             // DEAL_TIME
   char              type;             // DEAL_TYPE
   char              entry;            // DEAL_ENTRY
   string            symbol;           // DEAL_SYMBOL
   double            volume;            // DEAL_VOLUME
   double            price;             // DEAL_PRICE
   double            profit;            // DEAL_PROFIT
   double            swap;              // DEAL_SWAP
   double            commission;        // DEAL_COMMISSION
   long              magic;             // DEAL_MAGIC
   char              reason;            // DEAL_REASON
  };
//--- Structure to store the trade: the order of members corresponds to the position
struct Trade
  {
   datetime          time_in;           // entry time
   ulong             ticket;            // position ID
   char              type;              // buy or sell
   double            volume;            // volume
   string            symbol;            // symbol
   double            price_in;          // entry price
   datetime          time_out;          // exit time
   double            price_out;         // exit price
   double            commission;        // entry and exit commission
   double            swap;              // swap
   double            profit;            // profit or loss
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
   string filename=IntegerToString(AccountInfoInteger(ACCOUNT_LOGIN))+"_trades.sqlite"
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DAT
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- create the DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   int deals_total=HistoryDealsTotal();
   PrintFormat("Deals in the trading history: %d ", deals_total);
//--- add deals to the table
   if(!InsertDeals(db))
      return;
//--- show the first 10 deals
   Deal deals[], deal;
   ArrayResize(deals, 10);
   int request=DatabasePrepare(db, "SELECT * FROM DEALS");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   int i;
   for(i=0; DatabaseReadBind(request, deal); i++)
     {
      if(i>=10)
         break;
      deals[i].ticket=deal.ticket;
      deals[i].order_ticket=deal.order_ticket;
      deals[i].position_ticket=deal.position_ticket;
      deals[i].time=deal.time;
      deals[i].type=deal.type;
      deals[i].entry=deal.entry;
      deals[i].symbol=deal.symbol;
      deals[i].volume=deal.volume;
      deals[i].price=deal.price;
      deals[i].profit=deal.profit;
      deals[i].swap=deal.swap;
      deals[i].commission=deal.commission;
      deals[i].magic=deal.magic;
      deals[i].reason=deal.reason;
     }
//--- print the deals
   if(i>0)
     {
      ArrayResize(deals, i);
      PrintFormat("The first %d deals:", i);
      ArrayPrint(deals);
     }

//--- delete request after use
   DatabaseFinalize(request);

//--- make sure that hedging system for open position management is used on the accou
   if((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)!=ACCOUNT_MARG
     {
      //--- deals cannot be transformed to trades using a simple method through trans
      DatabaseClose(db);
      return;
     }

//--- now create the TRADES table based on the DEALS table
   if(!CreateTableTrades(db))
     {
      DatabaseClose(db);
      return;
     }
//--- fill in the TRADES table using an SQL query based on DEALS table data
   ulong start=GetMicrosecondCount();
   if(DatabaseTableExists(db, "DEALS"))
      //--- populate the TRADES table
      if(!DatabaseExecute(db, "INSERT INTO TRADES(TIME_IN,TICKET,TYPE,VOLUME,SYMBOL,P
                          "SELECT "
                          "   d1.time as time_in,"
                          "   d1.position_id as ticket,"
                          "   d1.type as type,"
                          "   d1.volume as volume,"
                          "   d1.symbol as symbol,"
                          "   d1.price as price_in,"
                          "   d2.time as time_out,"
                          "   d2.price as price_out,"
                          "   d1.commission+d2.commission as commission,"
                          "   d2.swap as swap,"
                          "   d2.profit as profit "
                          "FROM DEALS d1 "
                          "INNER JOIN DEALS d2 ON d1.position_id=d2.position_id "
                          "WHERE d1.entry=0 AND d2.entry=1     "))
        {
         Print("DB: fillng the TRADES table failed with code ", GetLastError());
         return;
        }
   ulong transaction_time=GetMicrosecondCount()-start;

//--- show the first 10 deals
   Trade trades[], trade;
   ArrayResize(trades, 10);
   request=DatabasePrepare(db, "SELECT * FROM TRADES");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   for(i=0; DatabaseReadBind(request, trade); i++)
     {
      if(i>=10)
```

```
         break;
      trades[i].time_in=trade.time_in;
      trades[i].ticket=trade.ticket;
      trades[i].type=trade.type;
      trades[i].volume=trade.volume;
      trades[i].symbol=trade.symbol;
      trades[i].price_in=trade.price_in;
      trades[i].time_out=trade.time_out;
      trades[i].price_out=trade.price_out;
      trades[i].commission=trade.commission;
      trades[i].swap=trade.swap;
      trades[i].profit=trade.profit;
     }
//--- print trades
   if(i>0)
     {
      ArrayResize(trades, i);
      PrintFormat("\r\nThe first %d trades:", i);
      ArrayPrint(trades);
      PrintFormat("Filling the TRADES table took %.2f milliseconds",double(transactio
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- close the database
   DatabaseClose(db);
  }
/*
Results:
   Deals in the trading history: 2741
   The first 10 deals:
       [ticket] [order_ticket] [position_ticket]              [time] [type] [entry] [s
   [0] 34429573              0                 0 2019.09.05 22:39:59      2       0 ""
   [1] 34432127       51447238          51447238 2019.09.06 06:00:03      0       0 "
   [2] 34432128       51447239          51447239 2019.09.06 06:00:03      1       0 "
   [3] 34432450       51447565          51447565 2019.09.06 07:00:00      0       0 "
   [4] 34432456       51447571          51447571 2019.09.06 07:00:00      1       0 "
   [5] 34432879       51448053          51448053 2019.09.06 08:00:00      1       0 "
   [6] 34432888       51448064          51448064 2019.09.06 08:00:00      0       0 "
   [7] 34435147       51450470          51450470 2019.09.06 10:30:00      1       0 "
   [8] 34435152       51450476          51450476 2019.09.06 10:30:00      0       0 "
   [9] 34435154       51450479          51450479 2019.09.06 10:30:00      1       0 "

   The first 10 trades:
                 [time_in] [ticket] [type] [volume] [symbol] [price_in]          [time
   [0] 2019.09.06 06:00:03 51447238      0  0.10000 "USDCAD"    1.32320 2019.09.06 18
   [1] 2019.09.06 06:00:03 51447239      1  0.10000 "USDCHF"    0.98697 2019.09.06 18
   [2] 2019.09.06 07:00:00 51447565      0  0.10000 "EURUSD"    1.10348 2019.09.09 03
   [3] 2019.09.06 07:00:00 51447571      1  0.10000 "AUDUSD"    0.68203 2019.09.09 03
   [4] 2019.09.06 08:00:00 51448053      1  0.10000 "USDCHF"    0.98701 2019.09.06 18:
   [5] 2019.09.06 08:00:00 51448064      0  0.10000 "USDJPY"  106.96200 2019.09.06 18:
   [6] 2019.09.06 10:30:00 51450470      1  0.10000 "EURUSD"    1.10399 2019.09.06 14:
   [7] 2019.09.06 10:30:00 51450476      0  0.10000 "GBPUSD"    1.23038 2019.09.06 14:
   [8] 2019.09.06 10:30:00 51450479      1  0.10000 "EURJPY"  118.12000 2019.09.06 14:
   [9] 2019.09.06 10:30:00 51450480      0  0.10000 "GBPJPY"  131.65300 2019.09.06 14:
   Filling the TRADES table took 12.51 milliseconds
*/
//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(database, "DEALS"))
     {
      return(false);
     }
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID          INT KEY NOT NULL,"
                          "ORDER_ID    INT     NOT NULL,"
                          "POSITION_ID INT     NOT NULL,"
                          "TIME        INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "ENTRY       INT     NOT NULL,"
                          "SYMBOL      CHAR(10),"
                          "VOLUME      REAL,"
                          "PRICE       REAL,"
                          "PROFIT      REAL,"
                          "SWAP        REAL,"
                          "COMMISSION  REAL,"
                          "MAGIC       INT,"
                          "REASON      INT );"))
        {
         Print("DB: create the DEALS table failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database       |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
```

```
     {
      Print("Failed to drop the DEALS table  with code ", GetLastError());
      return(false);
     }
//--- the table has been successfully deleted
   return(true);
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database)
  {
//--- Auxiliary variables
   ulong    deal_ticket;         // deal ticket
   long     order_ticket;        // the ticket of the order by which the deal was exec
   long     position_ticket;     // ID of the position to which the deal belongs
   datetime time;                // deal execution time
   long     type ;               // deal type
   long     entry ;              // deal direction
   string   symbol;              // the symbol fro which the deal was executed
   double   volume;              // operation volume
   double   price;               // price
   double   profit;              // financial result
   double   swap;                // swap
   double   commission;          // commission
   long     magic;               // Magic number (Expert Advisor ID)
   long     reason;              // deal execution reason or source
//--- go through all deals and add them to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
// --- lock the database before executing transactions
   DatabaseTransactionBegin(database);
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      //--- add each deal to the table using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TI
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G,
                                       deal_ticket, order_ticket, position_ticket, ti
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%d with code %d", __FUNCTION__, deal_
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
         break;
        }
     }
//--- check for transaction execution errors
   if(failed)
     {
      //--- roll back all transactions and unlock the database
      DatabaseTransactionRollback(database);
      PrintFormat("%s: DatabaseExecute() failed with code %d", __FUNCTION__, GetLastEr
      return(false);
     }
//--- all transactions have been performed successfully - record changes and unlock t
   DatabaseTransactionCommit(database);
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the TRADES table                                         |
//+------------------------------------------------------------------+
bool CreateTableTrades(int database)
  {
//--- if the TRADES table already exists, delete it
   if(!DeleteTable(database, "TRADES"))
      return(false);
//--- check if the table exists
   if(!DatabaseTableExists(database, "TRADES"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE TRADES("
                          "TIME_IN     INT     NOT NULL,"
                          "TICKET      INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "VOLUME      REAL,"
                          "SYMBOL      CHAR(10),"
                          "PRICE_IN    REAL,"
                          "TIME_OUT    INT     NOT NULL,"
                          "PRICE_OUT   REAL,"
                          "COMMISSION  REAL,"
                          "SWAP        REAL,"
                          "PROFIT      REAL);"))
        {
         Print("DB: create the TRADES table failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
```

See also

DatabaseExecute, DatabaseFinalize

DatabaseReset

Resets a request, like after calling DatabasePrepare().

```
int  DatabaseReset(
   int  request      // request handle received in DatabasePrepare
   );
```

Parameters

request

[in] The handle of the request obtained in DatabasePrepare().

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- SQLite error codes starting with ERR_DATABASE_ERROR(5601).

Note

The DatabaseReset() function is intended for multiple execution of a request with different parameter values. For example, when adding data to the table in bulk using the INSERT command, a custom set of field values should be formed for each entry.

Unlike DatabasePrepare(), the DatabaseReset() call does not compile the string with SQL commands into a new request, therefore DatabaseReset() is executed much faster than DatabasePrepare().

DatabaseReset() is used together with the DatabaseBind() functions and/or DatabaseBindArray() if the request parameter values should be changed after executing DatabaseRead(). This means that before setting new values of the request parameters (before the block of DatabaseBind/DatabaseBindArray calls), DatabaseReset() should be called to reset it. The parametrized request itself should be created using DatabasePrepare().

Just like DatabasePrepare(), DatabaseReset() does not make a database request. A direct request execution is performed when calling DatabaseRead() or DatabaseReadBind().

DatabaseReset() call does not lead to resetting parameter values in the request if they were set by calling DatabaseBind()/DatabaseBindArray(), i.e. the parameters retain their values. Thus, the value of only a single parameter can be changed. There is no need to set all request parameters anew after calling DatabaseReset().

A handle of a request removed using DatabaseFinalize() cannot be passed to DatabaseReset(). This will result in an error.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create or open a database
```

```
   string filename="symbols.sqlite";
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE);
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");
//--- if the SYMBOLS table exists, delete it
   if(DatabaseTableExists(db, "SYMBOLS"))
     {
      //--- delete the table
      if(!DatabaseExecute(db, "DROP TABLE SYMBOLS"))
        {
         Print("Failed to drop table SYMBOLS with code ", GetLastError());
         DatabaseClose(db);
         return;
        }
     }
//--- create the SYMBOLS table
   if(!DatabaseExecute(db, "CREATE TABLE SYMBOLS("
                       "NAME           TEXT    NOT NULL,"
                       "DESCRIPTION    TEXT            ,"
                       "PATH           TEXT            ,"
                       "SPREAD         INT             ,"
                       "POINT          REAL    NOT NULL,"
                       "DIGITS         INT     NOT NULL,"
                       "JSON           BLOB );"))
     {
      Print("DB: ", filename, " create table failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- display the list of all fields in the SYMBOLS table
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(SYMBOLS)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(SYMBOLS)\") failed, error code=%
      DatabaseClose(db);
      return;
     }

//--- create a parametrized request to add symbols to the SYMBOLS table
   string sql="INSERT INTO SYMBOLS (NAME,DESCRIPTION,PATH,SPREAD,POINT,DIGITS,JSON)"
              " VALUES (?1,?2,?3,?4,?5,?6,?7);"; // request parameters
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }

//--- go through all the symbols and add them to the SYMBOLS table
   int symbols=SymbolsTotal(false);
   bool request_error=false;
   DatabaseTransactionBegin(db);
   for(int i=0; i<symbols; i++)
     {
      //--- set the values of the parameters before adding a symbol
      ResetLastError();
      string symbol=SymbolName(i, false);
      if(!DatabaseBind(request, 0, symbol))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      //--- if the previous DatabaseBind() call was successful, set the next paramete
      if(!DatabaseBind(request, 1, SymbolInfoString(symbol, SYMBOL_DESCRIPTION)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 2, SymbolInfoString(symbol, SYMBOL_PATH)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 3, SymbolInfoInteger(symbol, SYMBOL_SPREAD)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 4, SymbolInfoDouble(symbol, SYMBOL_POINT)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 5, SymbolInfoInteger(symbol, SYMBOL_DIGITS)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 6, GetSymBolAsJson(symbol)))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, GetLas
         request_error=true;
         break;
        }

      //--- execute a request for inserting the entry and check for an error
      if(!DatabaseRead(request)&&(GetLastError()!=ERR_DATABASE_NO_MORE_DATA))
        {
         PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
      else
         PrintFormat("%d: added %s", i+1, symbol);
      //--- reset the request before the next parameter update
      if(!DatabaseReset(request))
        {
         PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
     } //--- done going through all the symbols

//--- transactions status
   if(request_error)
     {
      PrintFormat("Table SYMBOLS: failed to add %d symbols", symbols);
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table SYMBOLS: added %d symbols",symbols);
     }

//--- save the SYMBOLS table to a CSV file
   string csv_filename="symbols.csv";
   if(DatabaseExport(db, "SELECT * FROM SYMBOLS", csv_filename,
                     DATABASE_EXPORT_HEADER|DATABASE_EXPORT_INDEX|DATABASE_EXPORT_QUOT
      Print("Database: table SYMBOLS saved in ", csv_filename);
   else
```

```
      Print("Database: DatabaseExport(\"SELECT * FROM SYMBOLS\") failed with code", Ge

//--- close the database file and inform of that
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
  }
//+------------------------------------------------------------------+
//| Return a symbol specification as JSON                            |
//+------------------------------------------------------------------+
string GetSymBolAsJson(string symbol)
  {
//--- indents
   string indent1=Indent(1);
   string indent2=Indent(2);
   string indent3=Indent(3);
//---
   int digits=(int)SymbolInfoInteger(symbol, SYMBOL_DIGITS);
   string json="{"+
               "\n"+indent1+"\"ConfigSymbols\":["+
               "\n"+indent2+"{"+
               "\n"+indent3+"\"Symbol\":\""+symbol+"\","+
               "\n"+indent3+"\"Path\":\""+SymbolInfoString(symbol, SYMBOL_PATH)+"\","
               "\n"+indent3+"\"CurrencyBase\":\""+SymbolInfoString(symbol, SYMBOL_CUR
               "\n"+indent3+"\"CurrencyProfit\":\""+SymbolInfoString(symbol, SYMBOL_CU
               "\n"+indent3+"\"CurrencyMargin\":\""+SymbolInfoString(symbol, SYMBOL_CU
               "\n"+indent3+"\"ColorBackground\":\""+ColorToString((color)SymbolInfoI
               "\n"+indent3+"\"Digits\":\""+IntegerToString(SymbolInfoInteger(symbol,
               "\n"+indent3+"\"Point\":\""+DoubleToString(SymbolInfoDouble(symbol, SY
               "\n"+indent3+"\"TickBookDepth\":\""+IntegerToString(SymbolInfoInteger(
               "\n"+indent3+"\"ChartMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"TradeMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"TradeCalcMode\":\""+IntegerToString(SymbolInfoInteger(
               "\n"+indent3+"\"OrderMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"CalculationMode\":\""+IntegerToString(SymbolInfoIntege
               "\n"+indent3+"\"ExecutionMode\":\""+IntegerToString(SymbolInfoInteger(
               "\n"+indent3+"\"ExpirationMode\":\""+IntegerToString(SymbolInfoInteger
               "\n"+indent3+"\"FillFlags\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"ExpirFlags\":\""+IntegerToString(SymbolInfoInteger(symb
               "\n"+indent3+"\"Spread\":\""+IntegerToString(SymbolInfoInteger(symbol,
               "\n"+indent3+"\"TickValue\":\""+StringFormat("%G", (SymbolInfoDouble(sy
               "\n"+indent3+"\"TickSize\":\""+StringFormat("%G", (SymbolInfoDouble(sy
               "\n"+indent3+"\"ContractSize\":\""+StringFormat("%G",(SymbolInfoDouble
               "\n"+indent3+"\"StopsLevel\":\""+IntegerToString(SymbolInfoInteger(symb
               "\n"+indent3+"\"VolumeMin\":\""+StringFormat("%G",(SymbolInfoDouble(sy
               "\n"+indent3+"\"VolumeMax\":\""+StringFormat("%G",(SymbolInfoDouble(sy
               "\n"+indent3+"\"VolumeStep\":\""+StringFormat("%G",(SymbolInfoDouble(sy
               "\n"+indent3+"\"VolumeLimit\":\""+StringFormat("%G",(SymbolInfoDouble(
               "\n"+indent3+"\"SwapMode\":\""+IntegerToString(SymbolInfoInteger(symbo
               "\n"+indent3+"\"SwapLong\":\""+StringFormat("%G",(SymbolInfoDouble(symb
               "\n"+indent3+"\"SwapShort\":\""+StringFormat("%G",(SymbolInfoDouble(sy
               "\n"+indent3+"\"Swap3Day\":\""+IntegerToString(SymbolInfoInteger(symbol
               "\n"+indent2+"}"+
               "\n"+indent1+"]"+
               "\n}";
   return(json);
  }
//+-------------------------------------------------------------------+
//| Form an indent made of spaces                                     |
//+-------------------------------------------------------------------+
string Indent(const int number, const int characters=3)
  {
   int length=number*characters;
   string indent=NULL;
   StringInit(indent, length, ' ');
   return indent;
  }
/*
 Result:
  Database: symbols.sqlite opened successfully
  #| cid name        type notnull dflt_value pk
  -+-------------------------------------------
  1|   0 NAME        TEXT       1             0
  2|   1 DESCRIPTION TEXT       0             0
  3|   2 PATH        TEXT       0             0
  4|   3 SPREAD      INT        0             0
  5|   4 POINT       REAL       1             0
  6|   5 DIGITS      INT        1             0
  7|   6 JSON        BLOB       0             0
  1: added EURUSD
  2: added GBPUSD
  3: added USDCHF
  ...
  82: added USDCOP
  83: added USDARS
  84: added USDCLP
  Table SYMBOLS: added 84 symbols
  Database: table SYMBOLS saved in symbols.csv
  Database: symbols.sqlite created and closed
*/
```

See also

DatabasePrepare, DatabaseBind, DatabaseBindArray, DatabaseFinalize

DatabaseBind

Sets a parameter value in a request.

```
bool  DatabaseBind(
   int  request,      // the handle of a request created in DatabasePrepare
   int  index,        // the parameter index in the request
   T    value         // the value of a simple type parameter
   );
```

Parameters

request

[in] The handle of a request created in DatabasePrepare().

index

[in] The parameter index in the request a value should be set for. The numbering starts with zero.

value

[in] The value to be set. Extended types: bool, char, uchar, short, ushart, int, uint, color, datetime, long, ulong, float, double, string.

Return Value

Returns true if successful, otherwise - false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) - unsupported type;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_NOT_READY (5128) - cannot use the function to make a request at the moment. The request is being executed or already complete. DatabaseReset() should be called.

Note

The function is used in case an SQL request contains "?" or "?N" parameterizable values where N means the parameter index (starting from one). At the same time, parameters indexing in DatabaseBind() starts from zero.

For example:

```
INSERT INTO table VALUES (?,?,?)
SELECT * FROM table WHERE id=?
```

The function may be called immediately after a parametrized request is created in DatabasePrepare() or after the request is reset using DatabaseReset().

Use this function together with DatabaseReset() to execute the request as many times as needed with different parameter values.

The function is designed to work with simple type parameters. If a parameter should be checked against an array, use the DatabaseBindArray() function.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlTick ticks[];
//--- remember the start time before receiving the ticks
   uint start=GetTickCount();
//--- request the tick history per day
   ulong to=TimeCurrent()*1000;
   ulong from=to-PeriodSeconds(PERIOD_D1)*1000;
   if(CopyTicksRange(_Symbol, ticks, COPY_TICKS_ALL, from, to)==-1)
     {
      PrintFormat("%s: CopyTicksRange(%s - %s) failed, error=%d",
                  _Symbol, TimeToString(datetime(from/1000)), TimeToString(datetime(t
      return;
     }
   else
     {
      //--- how many ticks were received and how much time it took to receive them
      PrintFormat("%s: CopyTicksRange received %d ticks in %d ms (from %s to %s)",
                  _Symbol, ArraySize(ticks), GetTickCount()-start,
                  TimeToString(datetime(from/1000)), TimeToString(datetime(to/1000)));
     }

//--- set the file name for storing the database
   string filename=_Symbol+" "+TimeToString(datetime(from/1000))+" - "+TimeToString(da
   StringReplace(filename, ":", "."); // ":" character is not allowed in file names
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("Database: ", filename, " open failed with code ", GetLastError());
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");

//--- create the TICKS table
   if(!DatabaseExecute(db, "CREATE TABLE TICKS("
                       "SYMBOL             CHAR(10),"
                       "TIME               INT NOT NULL,"
                       "BID                REAL,"
                       "ASK                REAL,"
                       "LAST               REAL,"
                       "VOLUME             INT,"
                       "TIME_MSC           INT,"
                       "VOLUME_REAL        REAL);"))
     {
      Print("DB: ", filename, " create table TICKS failed with code ", GetLastError())
      DatabaseClose(db);
      return;
     }
//--- display the list of all fields in the TICKS table
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(TICKS)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(TICKS)\") failed, error code=%d a
      DatabaseClose(db);
      return;
     }
//--- create a parametrized request to add ticks to the TICKS table
   string sql="INSERT INTO TICKS (SYMBOL,TIME,BID,ASK,LAST,VOLUME,TIME_MSC,VOLUME_REA
              " VALUES (?1,?2,?3,?4,?5,?6,?7,?8)"; // request parameters
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }
//--- set the value of the first request parameter
   DatabaseBind(request, 0, _Symbol);
//--- remember the start time before adding ticks to the TICKS table
   start=GetTickCount();
   DatabaseTransactionBegin(db);
   int total=ArraySize(ticks);
   bool request_error=false;
   for(int i=0; i<total; i++)
     {
      //--- set the values of the remaining parameters before adding the entry
      ResetLastError();
      if(!DatabaseBind(request, 1, ticks[i].time))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      //--- if the previous DatabaseBind() call was successful, set the next paramete
      if(!request_error && !DatabaseBind(request, 2, ticks[i].bid))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 3, ticks[i].ask))
```

```
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 4, ticks[i].last))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 5, ticks[i].volume))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 6, ticks[i].time_msc))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }
      if(!request_error && !DatabaseBind(request, 7, ticks[i].volume_real))
        {
         PrintFormat("DatabaseBind() failed with code=%d", GetLastError());
         PrintFormat("Tick #%d line=%d", i+1, __LINE__);
         request_error=true;
         break;
        }

      //--- execute a request for inserting the entry and check for an error
      if(!request_error && !DatabaseRead(request) && (GetLastError()!=ERR_DATABASE_NO_
        {
         PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
      //--- reset the request before the next parameter update
      if(!request_error && !DatabaseReset(request))
        {
         PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
     } //--- done going through all the ticks

//--- transactions status
   if(request_error)
     {
      PrintFormat("Table TICKS: failed to add %d ticks ", ArraySize(ticks));
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table TICKS: added %d ticks in %d ms",
                  ArraySize(ticks), GetTickCount()-start);
     }

//--- close the database file and inform of that
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
  }
/*
 Result:
  EURUSD: CopyTicksRange received 268061 ticks in 47 ms (from 2020.03.18 12:40 to 202
  Database: EURUSD 2020.03.18 12.40 - 2020.03.19 12.40.sqlite opened successfully
  #| cid name        type     notnull dflt_value pk
  -+-------------------------------------------------
  1|   0 SYMBOL      CHAR(10)       0              0
  2|   1 TIME        INT            1              0
  3|   2 BID         REAL           0              0
  4|   3 ASK         REAL           0              0
  5|   4 LAST        REAL           0              0
  6|   5 VOLUME      INT            0              0
  7|   6 TIME_MSC    INT            0              0
  8|   7 VOLUME_REAL REAL           0              0
  Table TICKS: added 268061 ticks in 797 ms
  Database: EURUSD 2020.03.18 12.40 - 2020.03.19 12.40.sqlite created and closed
  OnCalculateCorrelation=0.87 2020.03.19 13:00:  EURUSD vs GBPUSD  PERIOD_M30
*/
```

See also

DatabasePrepare, DatabaseReset, DatabaseRead, DatabaseBindArray

DatabaseBindArray

Sets an array as a parameter value.

```
bool  DatabaseBind(
   int  request,       // the handle of a request created in DatabasePrepare
   int  index,         // the parameter index in the request
   T&   array[]        // parameter value as an array
   );
```

Parameters

request

[in] The handle of a request created in DatabasePrepare().

index

[in] The parameter index in the request a value should be set for. The numbering starts with zero.

array[]

[in] The array to be set as a request parameter value.

Return Value

Returns true if successful, otherwise - false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) - unsupported type;
- ERR_ARRAY_BAD_SIZE (4011) - array size in bytes exceeds INT_MAX;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_NOT_READY (5128) - cannot use the function to make a request at the moment (the request is being executed or already complete, DatabaseReset should be called).

Note

The function is used in case an SQL request contains "?" or "?N" parameterizable values where N means the parameter index (starting from one). At the same time, parameters indexing in DatabaseBindArray() starts from zero.

For example:

```
INSERT INTO table VALUES (?,?,?)
```

The function may be called immediately after a parametrized request is created in DatabasePrepare() or after the request is reset using DatabaseReset().

Use this function together with DatabaseReset() to execute the request as many times as needed with different parameter values.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- open the dialog for selecting files with the DAT extension
   string selected_files[];
   if(!FileSelectDialog("Select files to download", NULL,
                       "Data files (*.dat)|*.dat|All files (*.*)|*.*",
                       FSD_ALLOW_MULTISELECT, selected_files, "tester.dat")>0)
     {
      Print("Files not selected. Exit");
      return;
     }
//--- get the size of files
   ulong filesize[];
   int filehandle[];
   int files=ArraySize(selected_files);
   ArrayResize(filesize, files);
   ZeroMemory(filesize);
   ArrayResize(filehandle, files);
   double total_size=0;
   for(int i=0; i<files; i++)
     {
      filehandle[i]=FileOpen(selected_files[i], FILE_READ|FILE_BIN);
      if(filehandle[i]!=INVALID_HANDLE)
        {
         filesize[i]=FileSize(filehandle[i]);
         //PrintFormat("%d, %s handle=%d %d bytes", i, selected_files[i], filehandle[
         total_size+=(double)filesize[i];
        }
     }
//--- check the common size of files
   if(total_size==0)
     {
      PrintFormat("Total files size is 0. Exit");
      return;
     }

//--- create or open the database in the common terminal folder
   string filename="dat_files.sqlite";
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE);
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
   else
      Print("Database: ", filename, " opened successfully");
//--- if the FILES table exists, delete it
   if(DatabaseTableExists(db, "FILES"))
     {
      //--- delete the table
      if(!DatabaseExecute(db, "DROP TABLE FILES"))
        {
         Print("Failed to drop table FILES with code ", GetLastError());
         DatabaseClose(db);
         return;
        }
     }
//--- create the FILES table
   if(!DatabaseExecute(db, "CREATE TABLE FILES("
                       "NAME           TEXT NOT NULL,"
                       "SIZE           INT  NOT NULL,"
                       "PERCENT_SIZE   REAL NOT NULL,"
                       "DATA           BLOB NOT NULL);"))
     {
      Print("DB: failed to create table FILES with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- display the list of all fields in the FILES table
   if(DatabasePrint(db, "PRAGMA TABLE_INFO(FILES)", 0)<0)
     {
      PrintFormat("DatabasePrint(\"PRAGMA TABLE_INFO(FILES)\") failed, error code=%d
      DatabaseClose(db);
      return;
     }

//--- create a parametrized request to add files to the FILES table
   string sql="INSERT INTO FILES (NAME,SIZE,PERCENT_SIZE,DATA)"
              " VALUES (?1,?2,?3,?4);"; // request parameters
   int request=DatabasePrepare(db, sql);
   if(request==INVALID_HANDLE)
     {
      PrintFormat("DatabasePrepare() failed with code=%d", GetLastError());
      Print("SQL request: ", sql);
      DatabaseClose(db);
      return;
     }

//--- go through all the files and add them to the FILES table
   bool request_error=false;
   DatabaseTransactionBegin(db);
   int count=0;
   uint size;
   for(int i=0; i<files; i++)
     {
      if(filehandle[i]!=INVALID_HANDLE)
        {
         char data[];
         size=FileReadArray(filehandle[i], data);
      if(size==0)
        {
         PrintFormat("FileReadArray(%s) failed with code %d", selected_files[i], Ge
         continue;
        }

      count++;
      //--- set the values of the parameters before adding the file to the table
      if(!DatabaseBind(request, 0, selected_files[i]))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 1, size))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBind(request, 2, double(size)*100./total_size))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      if(!DatabaseBindArray(request, 3, data))
        {
         PrintFormat("DatabaseBind() failed at line %d with code=%d", __LINE__, Get
         request_error=true;
         break;
        }
      //--- execute a request for inserting the entry and check for an error
      if(!DatabaseRead(request)&&(GetLastError()!=ERR_DATABASE_NO_MORE_DATA))
        {
         PrintFormat("DatabaseRead() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
        }
      else
         PrintFormat("%d. %s: %d bytes", count, selected_files[i],size);
      //--- reset the request before the next parameter update
      if(!DatabaseReset(request))
        {
         PrintFormat("DatabaseReset() failed with code=%d", GetLastError());
         DatabaseFinalize(request);
         request_error=true;
         break;
```

```
            }
         }
      }
//--- transactions status
   if(request_error)
     {
      PrintFormat("Table FILES: failed to add %d files", count);
      DatabaseTransactionRollback(db);
      DatabaseClose(db);
      return;
     }
   else
     {
      DatabaseTransactionCommit(db);
      PrintFormat("Table FILES: added %d files", count);
     }

//--- close the database file and inform of that
   DatabaseClose(db);
   PrintFormat("Database: %s created and closed", filename);
  }
```

See also

DatabasePrepare, DatabaseReset, DatabaseRead, DatabaseBind

DatabaseRead

Moves to the next entry as a result of a request.

```
bool  DatabaseRead(
   int   request      // request handle received in DatabasePrepare
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) – no table name specified (empty string or NULL);
- ERR_WRONG_STRING_PARAMETER (5040) – error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) – internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) – invalid database handle;
- ERR_DATABASE_EXECUTE (5124) – request execution error;
- ERR_DATABASE_NO_MORE_DATA (5126) – no table exists (not an error, normal completion).

See also

DatabasePrepare, DatabaseReadBind

DatabaseReadBind

Moves to the next record and reads data into the structure from it.

```
bool  DatabaseReadBind(
   int     request,           // the handle of a request created in DatabasePrepare
   void&   struct_object      // the reference to the structure for reading the record
   );
```

Parameters

request

[in] The handle of a request created in DatabasePrepare().

struct_object

[out] The reference to the structure the data from the current record is to be read to. The structure should only have numerical types and/or strings (arrays are not allowed) as members and cannot be a descendant.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INVALID_PARAMETER (4003) - no table name specified (empty string or NULL);
- ERR_WRONG_STRING_PARAMETER (5040) - error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_EXECUTE (5124) - request execution error;
- ERR_DATABASE_NO_MORE_DATA (5126) - no table exists (not an error, normal completion).

Note

A number of fields in the *struct_object* structure should not exceed DatabaseColumnsCount(). If the number of fields in the *struct_object* structure is less than the number of fields in the record, the partial reading is performed. The remaining data can be explicitly obtained using the corresponding DatabaseColumnText(), DatabaseColumnInteger() and other functions.

Example:

```
struct Person
  {
   int                id;
   string             name;
   int                age;
   string             address;
   double             salary;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
   int db;
   string filename="company.sqlite";
//--- open
   db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE |DATABASE_
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- if the table COMPANY exists then drop the table
   if(DatabaseTableExists(db, "COMPANY"))
     {
      //--- delete the table
      if(!DatabaseExecute(db, "DROP TABLE COMPANY"))
        {
         Print("Failed to drop table COMPANY with code ", GetLastError());
         DatabaseClose(db);
         return;
        }
     }
//--- create table
   if(!DatabaseExecute(db, "CREATE TABLE COMPANY("
                       "ID INT PRIMARY KEY     NOT NULL,"
                       "NAME           TEXT    NOT NULL,"
                       "AGE            INT     NOT NULL,"
                       "ADDRESS        CHAR(50),"
                       "SALARY         REAL );"))
     {
      Print("DB: ", filename, " create table failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }

//--- insert data
   if(!DatabaseExecute(db, "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, '
                       "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, '
     {
      Print("DB: ", filename, " insert failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }

//--- prepare the request
   int request=DatabasePrepare(db, "SELECT * FROM COMPANY WHERE SALARY>15000");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
//--- print records
   Person person;
   Print("Persons with salary > 15000:");
   for(int i=0; DatabaseReadBind(request, person); i++)
      Print(i, ":  ", person.id, " ", person.name, " ", person.age, " ", person.addres
//--- delete request after use
   DatabaseFinalize(request);

   Print("Some statistics:");
//--- prepare new request about total salary
   request=DatabasePrepare(db, "SELECT SUM(SALARY) FROM COMPANY");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      DatabaseClose(db);
      return;
     }
   while(DatabaseRead(request))
     {
      double total_salary;
      DatabaseColumnDouble(request, 0, total_salary);
      Print("Total salary=", total_salary);
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- prepare new request about average salary
   request=DatabasePrepare(db, "SELECT AVG(SALARY) FROM COMPANY");
   if(request==INVALID_HANDLE)
     {
      Print("DB: ", filename, " request failed with code ", GetLastError());
      ResetLastError();
      DatabaseClose(db);
      return;
     }
   while(DatabaseRead(request))
     {
      double aver_salary;
      DatabaseColumnDouble(request, 0, aver_salary);
      Print("Average salary=", aver_salary);
     }
//--- delete request after use
   DatabaseFinalize(request);

//--- close database
```

```
   DatabaseClose(db);
  }
//+------------------------------------------------------------------
/*
Output:
Persons with salary > 15000:
0:  1 Paul 32 California 25000.0
1:  3 Teddy 23 Norway 20000.0
2:  4 Mark 25 Rich-Mond  65000.0
Some statistics:
Total salary=125000.0
Average salary=31250.0
*/
```

See also

[DatabasePrepare](), [DatabaseRead]()

DatabaseFinalize

Removes a request created in DatabasePrepare().

```
void  DatabaseFinalize(
   int   request       // request handle received in DatabasePrepare
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

Return Value

None.

Note

If the handle is invalid, the function sets the ERR_DATABASE_INVALID_HANDLE error. You can check the error using GetLastError().

See also

DatabasePrepare, DatabaseExecute

DatabaseTransactionBegin

Starts transaction execution.

```
bool  DatabaseTransactionBegin(
   int  database      // database handle received in DatabaseOpen
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) - critical runtime error;
- ERR_INVALID_PARAMETER (4003) - sql parameter contains an empty string;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_WRONG_STRING_PARAMETER (5040) - error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_EXECUTE (5124) - request execution error.

Note

The DatabaseTransactionBegin() function should be called before a transaction execution. Any transaction should start with calling DatabaseTransactionBegin() and end with calling DatabaseTransactionCommit().

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create the file name
   string filename=AccountInfoString(ACCOUNT_SERVER) +"_"+IntegerToString(AccountInfoI
//--- open/create the database in the common terminal folder
   int db=DatabaseOpen(filename, DATABASE_OPEN_READWRITE | DATABASE_OPEN_CREATE | DATA
   if(db==INVALID_HANDLE)
     {
      Print("DB: ", filename, " open failed with code ", GetLastError());
      return;
     }
//--- if the DEALS table already exists, delete it
   if(!DeleteTable(db, "DEALS"))
     {
      DatabaseClose(db);
      return;
     }
```

```
//--- create the DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }
//---  request the entire trading history
   datetime from_date=0;
   datetime to_date=TimeCurrent();
//--- request the history of deals in the specified interval
   HistorySelect(from_date, to_date);
   int deals_total=HistoryDealsTotal();
   PrintFormat("Deals in the trading history: %d ", deals_total);

//--- measure the transaction execution speed using DatabaseTransactionBegin/Database
   ulong start=GetMicrosecondCount();
   bool fast_transactions=true;
   InsertDeals(db, fast_transactions);
   double fast_transactions_time=double(GetMicrosecondCount()-start)/1000;
   PrintFormat("Transations WITH    DatabaseTransactionBegin/DatabaseTransactionCommit

//--- delete the DEALS table, and then create it again
   if(!DeleteTable(db, "DEALS"))
     {
      DatabaseClose(db);
      return;
     }
//--- create a new DEALS table
   if(!CreateTableDeals(db))
     {
      DatabaseClose(db);
      return;
     }

//--- test again, this time without using DatabaseTransactionBegin/DatabaseTransactio
   fast_transactions=false;
   start=GetMicrosecondCount();
   InsertDeals(db, fast_transactions);
   double slow_transactions_time=double(GetMicrosecondCount()-start)/1000;
   PrintFormat("Transations WITHOUT DatabaseTransactionBegin/DatabaseTransactionCommit
//--- report gain in time
   PrintFormat("Use of DatabaseTransactionBegin/DatabaseTransactionCommit provided ac
//--- close the database
   DatabaseClose(db);
  }
/*
Results:
   Deals in the trading history: 2737
   Transations WITH    DatabaseTransactionBegin/DatabaseTransactionCommit: time=48.5
   Transations WITHOUT DatabaseTransactionBegin/DatabaseTransactionCommit: time=25818
   Use of DatabaseTransactionBegin/DatabaseTransactionCommit provided acceleration by
*/
//+------------------------------------------------------------------+
//| Deletes a table with the specified name from the database        |
//+------------------------------------------------------------------+
bool DeleteTable(int database, string table_name)
  {
   if(!DatabaseExecute(database, "DROP TABLE IF EXISTS "+table_name))
     {
      Print("Failed to drop table with code ", GetLastError());
      return(false);
     }
//--- the table has been successfully deleted
   return(true);
  }
//+------------------------------------------------------------------+
//| Creates the DEALS table                                          |
//+------------------------------------------------------------------+
bool CreateTableDeals(int database)
  {
//--- check if the table exists
   if(!DatabaseTableExists(database, "DEALS"))
      //--- create the table
      if(!DatabaseExecute(database, "CREATE TABLE DEALS("
                          "ID          INT KEY NOT NULL,"
                          "ORDER_ID    INT     NOT NULL,"
                          "POSITION_ID INT     NOT NULL,"
                          "TIME        INT     NOT NULL,"
                          "TYPE        INT     NOT NULL,"
                          "ENTRY       INT     NOT NULL,"
                          "SYMBOL      CHAR(10),"
                          "VOLUME      REAL,"
                          "PRICE       REAL,"
                          "PROFIT      REAL,"
                          "SWAP        REAL,"
                          "COMMISSION  REAL,"
                          "MAGIC       INT,"
                          "REASON      INT );"))
        {
         Print("DB: create the table DEALS failed with code ", GetLastError());
         return(false);
        }
//--- the table has been successfully created
   return(true);
  }
//+------------------------------------------------------------------+
//| Adds deals to the database table                                 |
//+------------------------------------------------------------------+
bool InsertDeals(int database, bool begintransaction=true)
  {
//--- Auxiliary variables
   ulong    deal_ticket;         // deal ticket
   long     order_ticket;        // the ticket of the order by which the deal was exec
   long     position_ticket;     // ID of the position to which the deal belongs
   datetime time;                // deal execution time
   long     type ;               // deal type
   long     entry ;              // deal direction
   string   symbol;              // the symbol fro which the deal was executed
   double   volume;              // operation volume
   double   price;               // price
   double   profit;              // financial result
   double   swap;                // swap
   double   commission;          // commission
   long     magic;               // Magic number
   long     reason;              // deal execution reason or source
//--- go through all deals and add to the database
   bool failed=false;
   int deals=HistoryDealsTotal();
//--- if fast transaction performance method is used
   if(begintransaction)
     {
      // --- lock the database before executing transactions
      DatabaseTransactionBegin(database);
     }
   for(int i=0; i<deals; i++)
     {
      deal_ticket=    HistoryDealGetTicket(i);
      order_ticket=   HistoryDealGetInteger(deal_ticket, DEAL_ORDER);
      position_ticket=HistoryDealGetInteger(deal_ticket, DEAL_POSITION_ID);
      time= (datetime)HistoryDealGetInteger(deal_ticket, DEAL_TIME);
      type=           HistoryDealGetInteger(deal_ticket, DEAL_TYPE);
      entry=          HistoryDealGetInteger(deal_ticket, DEAL_ENTRY);
      symbol=         HistoryDealGetString(deal_ticket, DEAL_SYMBOL);
      volume=         HistoryDealGetDouble(deal_ticket, DEAL_VOLUME);
      price=          HistoryDealGetDouble(deal_ticket, DEAL_PRICE);
      profit=         HistoryDealGetDouble(deal_ticket, DEAL_PROFIT);
      swap=           HistoryDealGetDouble(deal_ticket, DEAL_SWAP);
      commission=     HistoryDealGetDouble(deal_ticket, DEAL_COMMISSION);
      magic=          HistoryDealGetInteger(deal_ticket, DEAL_MAGIC);
      reason=         HistoryDealGetInteger(deal_ticket, DEAL_REASON);
      //--- add each deal using the following request
      string request_text=StringFormat("INSERT INTO DEALS (ID,ORDER_ID,POSITION_ID,TI
                                       "VALUES (%d, %d, %d, %d, %d, %d, '%s', %G, %G,
                                       deal_ticket, order_ticket, position_ticket, ti
      if(!DatabaseExecute(database, request_text))
        {
         PrintFormat("%s: failed to insert deal #%dwith code %d", __FUNCTION__, deal_t
         PrintFormat("i=%d: deal #%d  %s", i, deal_ticket, symbol);
         failed=true;
         break;
        }
     }
//--- check for transaction execution errors
   if(failed)
     {
      //--- if fast transaction performance method is used
      if(begintransaction)
        {
         //--- roll back all transactions and unlock the database
         DatabaseTransactionRollback(database);
        }
      Print("%s: DatabaseExecute() failed with code ", __FUNCTION__, GetLastError());
      return(false);
     }
//--- if fast transaction performance method is used
   if(begintransaction)
     {
      //--- all transactions have been performed successfully - record changes and unl
      DatabaseTransactionCommit(database);
     }
//--- successful completion
   return(true);
  }
//+------------------------------------------------------------------+
```

See also

[DatabaseExecute](), [DatabasePrepare](), [DatabaseTransactionCommit](), [DatabaseTransactionRollback]()

DatabaseTransactionCommit

Completes transaction execution.

```
bool  DatabaseTransactionCommit(
   int   database       // database handle received in DatabaseOpen
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

ERR_INTERNAL_ERROR (4001)	critical runtime error;
ERR_INVALID_PARAMETER (4003)	sql parameter contains an empty string;
ERR_NOT_ENOUGH_MEMORY (4004)	insufficient memory;
ERR_WRONG_STRING_PARAMETER (5040)	error converting a request into a UTF-8 string;
ERR_DATABASE_INTERNAL (5120)	internal database error;
ERR_DATABASE_INVALID_HANDLE (5121)	invalid database handle;
ERR_DATABASE_EXECUTE (5124)	request execution error.

Note

The DatabaseTransactionCommit() function completes all transactions executed after calling the DatabaseBeginTransaction() function. Any transaction should start with calling DatabaseTransactionBegin() and end with calling DatabaseTransactionCommit() for successful completion.

See also

DatabaseExecute, DatabasePrepare, DatabaseTransactionBegin, DatabaseTransactionRollback

DatabaseTransactionRollback

Rolls back transactions.

```
bool  DatabaseTransactionRollback(
   int  database      // database handle received in DatabaseOpen
   );
```

Parameters

database

[in] Database handle received in DatabaseOpen().

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_INTERNAL_ERROR (4001) - critical runtime error;
- ERR_INVALID_PARAMETER (4003) - sql parameter contains an empty string;
- ERR_NOT_ENOUGH_MEMORY (4004) - insufficient memory;
- ERR_WRONG_STRING_PARAMETER (5040) - error converting a request into a UTF-8 string;
- ERR_DATABASE_INTERNAL (5120) - internal database error;
- ERR_DATABASE_INVALID_HANDLE (5121) - invalid database handle;
- ERR_DATABASE_EXECUTE (5124) - request execution error.

Note

DatabaseTransactionRollback() call cancels all transactions executed after calling the DatabaseTransactionBegin() function. The DatabaseTransactionRollback() function is necessary for rolling back changes in a database in case errors occur when executing a transaction.

See also

DatabaseExecute, DatabasePrepare, DatabaseTransactionBegin, DatabaseTransactionCommit

DatabaseColumnsCount

Gets the number of fields in a request.

```
int  DatabaseColumnsCount(
   int  request      // request handle received in DatabasePrepare
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

Return Value

Number of fields or -1 in case of an error. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) - invalid request handle.

Note

There is no need to call the DatabaseRead() function to get the number of fields of a request created in DatabasePrepare(). For the remaining DatabaseColumnXXX() functions, DatabaseRead() should be preliminarily called.

See also

DatabasePrepare, DatabaseFinalize, DatabaseClose

DatabaseColumnName

Gets a field name by index.

```
bool  DatabaseColumnName(
   int       request,      // request handle received in DatabasePrepare
   int       column,       // field index in the request
   string&   name          // the reference to the variable for receiving the field name
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

name

[out] Variable for writing the field name.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) - invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) - 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnType

DatabaseColumnType

Gets a field type by index.

```
ENUM_DATABASE_FIELD_TYPE  DatabaseColumnType(
   int  request,      // request handle received in DatabasePrepare
   int  column        // field index in the request
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

Return Value

Return the field type from the ENUM_DATABASE_FIELD_TYPE enumeration. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) – invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) – 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

ENUM_DATABASE_FIELD_TYPE

ID	Description
DATABASE_FIELD_TYPE_INVALID	Error getting type, the error code can be obtained using int GetLastError()
DATABASE_FIELD_TYPE_INTEGER	Integer type
DATABASE_FIELD_TYPE_FLOAT	Real type
DATABASE_FIELD_TYPE_TEXT	String type
DATABASE_FIELD_TYPE_BLOB	Binary type
DATABASE_FIELD_TYPE_NULL	Special NULL type

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnName

DatabaseColumnSize

Gets a field size in bytes.

```
int  DatabaseColumnSize(
   int  request,      // request handle received in DatabasePrepare
   int  column        // field index in the request
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

Return Value

If successful, the field size in bytes is returned, otherwise -1. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) – invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) – 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

See also

DatabasePrepare, DatabaseColumnBlob, DatabaseColumnsCount, DatabaseColumnName, DatabaseColumnType

DatabaseColumnText

Gets a field value as a string from the current record.

```
bool  DatabaseColumnText(
   int       request,      // request handle received in DatabasePrepare
   int       column,       // field index in the request
   string&   value         // the reference to the variable for receiving the value
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

value

[out] Reference to the variable for writing the field value.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) - invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) - 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

To read the value from the next record, call DatabaseRead() preliminarily.

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnType, DatabaseColumnName

DatabaseColumnInteger

Gets the int type value from the current record.

```
bool  DatabaseColumnInteger(
   int    request,      // request handle received in DatabasePrepare
   int    column,       // field index in the request
   int&   value         // the reference to the variable for receiving the value
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

value

[out] Reference to the variable for writing the field value.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) – invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) – 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

To read the value from the next record, call DatabaseRead() preliminarily.

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnType, DatabaseColumnName

DatabaseColumnLong

Gets the long type value from the current record.

```
bool  DatabaseColumnLong(
   int     request,      // request handle received in DatabasePrepare
   int     column,       // field index in the request
   long&   value         // the reference to the variable for receiving the value
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

value

[out] Reference to the variable for writing the field value.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) – invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) – 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

To read the value from the next record, call DatabaseRead() preliminarily.

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnType, DatabaseColumnName

DatabaseColumnDouble

Gets the double type value from the current record.

```
bool  DatabaseColumnDouble(
   int       request,      // request handle received in DatabasePrepare
   int       column,       // field index in the request
   double&   value         // the reference to the variable for receiving the value
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

value

[out] Reference to the variable for writing the field value.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) – invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) – 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

To read the value from the next record, call DatabaseRead() preliminarily.

See also

DatabasePrepare, DatabaseColumnsCount, DatabaseColumnType, DatabaseColumnName

DatabaseColumnBlob

Gets a field value as an array from the current record.

```
bool  DatabaseColumnBlob(
   int     request,      // request handle received in DatabasePrepare
   int     column,       // field index in the request
   void&   data[]        // the reference to the variable for receiving the value
   );
```

Parameters

request

[in] Request handle received in DatabasePrepare().

column

[in] Field index in the request. Field numbering starts from zero and cannot exceed DatabaseColumnsCount() - 1.

data[]

[out] Reference to the array for writing the field value.

Return Value

Return true if successful, otherwise false. To get the error code, use GetLastError(), the possible responses are:

- ERR_DATABASE_INVALID_HANDLE (5121) - invalid request handle;
- ERR_DATABASE_NO_MORE_DATA (5126) - 'column' index exceeds DatabaseColumnsCount() -1.

Note

The value can be obtained only if at least one DatabaseRead() call has been preliminarily made for 'request'.

To read the value from the next record, call DatabaseRead() preliminarily.

See also

DatabasePrepare, DatabaseColumnSize, DatabaseColumnsCount, DatabaseColumnType, DatabaseColumnName

Working with DirectX

DirectX 11 functions and shaders are designed for 3D visualization directly on a price chart.

Creating 3D graphics requires a graphic context (DXContextCreate) with the necessary image size. Besides, it is necessary to prepare vertex and index buffers (DXBufferCreate), as well as create vertex and pixel shaders (DXShaderCreate). This is enough to display graphics in color.

The next level of graphics requires the inputs (DXInputSet) for passing additional rendering parameters to shaders. This allows setting the camera and 3D object positions, describe light sources and implement mouse and keyboard control.

Thus, the built-in MQL5 functions enable you to create animated 3D charts directly in MetaTrader 5 with no need for third-party tools. A video card should support DX 11 and Shader Model 5.0 for the functions to work.

To start working with the library, simply read the article How to create 3D graphics using DirectX in MetaTrader 5.

Function	Action
DXContextCreate	Creates a graphic context for rendering frames of a specified size
DXContextSetSize	Changes a frame size of a graphic context created in DXContextCreate()
DXContextSetSize	Gets a frame size of a graphic context created in DXContextCreate()
DXContextClearColors	Sets a specified color to all pixels for the rendering buffer
DXContextClearDepth	Clears the depth buffer
DXContextGetColors	Gets an image of a specified size and offset from a graphic context
DXContextGetDepth	Gets the depth buffer of a rendered frame
DXBufferCreate	Creates a buffer of a specified type based on a data array
DXTextureCreate	Creates a 2D texture out of a rectangle of a specified size cut from a passed image
DXInputCreate	Creates shader inputs
DXInputSet	Sets shader inputs
DXShaderCreate	Creates a shader of a specified type
DXShaderSetLayout	Sets vertex layout for the vertex shader
DXShaderInputsSet	Sets shader inputs
DXShaderTexturesSet	Sets shader textures
DXDraw	Renders the vertices of the vertex buffer set in DXBufferSet()

Function	Action
DXDrawIndexed	Renders graphic primitives described by the index buffer from DXBufferSet()
DXPrimiveTopologySet	Sets the type of primitives for rendering using DXDrawIndexed()
DXBufferSet	Sets a buffer for the current rendering
DXShaderSet	Sets a shader for rendering
DXHandleType	Returns a handle type
DXRelease	Releases a handle

DXContextCreate

Creates a graphic context for rendering frames of a specified size.

```
int  DXContextCreate(
   uint  width,       // width in pixels
   uint  height       // height in pixels
   );
```

Parameters

width

[in] Frame width in pixels.

height

[in] Frame height in pixels.

Return Value

A handle for a created context or INVALID_HANDLE in case of an error. To receive an error code, the GetLastError() function should be called.

Note

All graphical objects created using the DXBufferCreate, DXInputCreate, DXShaderCreate and DXTextureCreate functions can be used only in a graphic context they were created in.

A frame size can subsequently be changed to DXContextSetSize().

A created handle that is no longer in use should be explicitly released by the DXRelease() function.

DXContextSetSize

Changes a frame size of a graphic context created in DXContextCreate().

```
bool  DXContextSetSize(
   int      context,       // graphic context handle
   uint&    width,         // width in pixels
   uint&    height         // height in pixels
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

width

[in] Frame width in pixels.

height

[in] Frame height in pixels.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

A frame size of a graphic context should be changed only between frame renderings.

DXContextGetSize

Gets a frame size of a graphic context created in DXContextCreate().

```
bool  DXContextGetSize(
   int      context,       // graphic context handle
   uint&    width,         // width in pixels
   uint&    height         // height in pixels
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

width

[out] Frame width in pixels.

height

[out] Frame height in pixels.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

DXContextClearColors

Sets a specified color to all pixels for the rendering buffer.

```
bool  DXContextClearColors(
   int                 context,       // graphic context handle
   const DXVector&   color          // color
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

color

[in] Rendering color.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The DXContextClearColors() function can be used for clearing the color buffer before rendering the next frame.

DXContextClearDepth

Clears the depth buffer.

```
bool  DXContextClearDepth(
   int  context       // graphic context handle
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The DXContextClearDepth() function can be used for clearing the depth buffer before rendering the next frame.

DXContextGetColors

Gets an image of a specified size and offset from a graphic context.

```
bool  DXContextGetColors(
   int     context,                     // graphic context handle
   uint&   image[],                     // image pixels array
   int     image_width=WHOLE_ARRAY,     // image width in pixels
   int     image_height=WHOLE_ARRAY,    // image height in pixels
   int     image_offset_x=0,            // X offset
   int     image_offset_y=0             // Y offset
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

image

[out] The array of *image_width**image_height* pixels in ARGB format.

image_width=WHOLE_ARRAY

[in] Image width in pixels.

image_height=WHOLE_ARRAY

[in] Image height in pixels.

image_offset_x=0

[in] X offset.

image_offset_y=0

[in] Y offset.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

DXContextGetDepth

Gets the depth buffer of a rendered frame.

```
bool  DXContextGetDepth(
   int      context,       // graphic context handle
   float&   image[]        // depth value array
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

image

[out] Array of the rendered frame depth buffer values.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The returned buffer contains the depth of each pixel of a rendered frame that can be obtained in DXContextGetColors() in relative units (from 0.0 to 1.0).

DXBufferCreate

Creates a buffer of a specified type based on a data array.

```
int  DXBufferCreate(
   int                     context,               // graphic context handle
   ENUM_DX_BUFFER_TYPE     buffer_type,           // type of a created buffer
   const void&             data[],                // buffer data
   uint                    start=0,               // initial index
   uint                    count=WHOLE_ARRAY      // number of elements
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

buffer_type

[in] Buffer type from the ENUM_DX_BUFFER_TYPE enumeration.

data[]

[in] Data for creating a buffer.

start=0

[in] Index of the first element of the array, starting from which the array values are used to create a buffer. By default, the data is taken from the beginning of the array.

count=WHOLE_ARRAY

[in] Number of values. By default, the entire array is used (count=WHOLE_ARRAY).

Return Value

The handle for a created buffer or INVALID_HANDLE in case of an error. To receive an error code, the GetLastError() function should be called.

Note

For the index buffer, the *data[]* array should be of 'uint' type, while the vertex buffer receives the array of structures describing vertices.

A created handle that is no longer in use should be explicitly released by the DXRelease() function.

ENUM_DX_BUFFER_TYPE

ID	Value	Description
DX_BUFFER_VERTEX	1	Vertex buffer
DX_BUFFER_INDEX	2	Index buffer

DXTextureCreate

Creates a 2D texture out of a rectangle of a specified size cut from a passed image.

```
int  DXTextureCreate(
   int                context,         // graphic context handle
   ENUM_DX_FORMAT     format,          // pixel color format
   uint               width,           // source image width
   uint               height,          // source image height
   const void&        data[],          // array of source image pixels
   uint               data_x,          // X coordinate of a rectangle used to create a te
   uint               data_y,          // Y coordinate of a rectangle used to create a te
   uint               data_width,      // rectangle width for creating a texture
   uint               data_height      // rectangle height for creating a texture
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

format

[in] Pixel color format set from the ENUM_DX_FORMAT enumeration.

width

[in] Width of an image a texture is based on.

height

[in] Height of an image a texture is based on.

data

[in] Pixel array of an image a texture is based on.

data_x

[in] X coordinate of a rectangle (X offset) used to create a texture.

data_y

[in] Y coordinate of a rectangle (Y offset) used to create a texture.

data_width

[in] Width of a rectangle used to create a texture.

data_height

[in] Height of a rectangle used to create a texture.

Return Value

Texture handle or INVALID_HANDLE in case of an error. To receive an error code, the GetLastError() function should be called.

Note

A created handle that is no longer in use should be explicitly released by the DXRelease() function.

ENUM_DX_FORMAT

ID	Value	Match in DXGI_FORMAT
DX_FORMAT_UNKNOWN	0	DXGI_FORMAT_UNKNOWN
DX_FORMAT_R32G32B32A32_TYPELESS	1	DXGI_FORMAT_R32G32B32A32_TYPELESS
DX_FORMAT_R32G32B32A32_FLOAT	2	DXGI_FORMAT_R32G32B32A32_FLOAT
DX_FORMAT_R32G32B32A32_UINT	3	DXGI_FORMAT_R32G32B32A32_UINT
DX_FORMAT_R32G32B32A32_SINT	4	DXGI_FORMAT_R32G32B32A32_SINT
DX_FORMAT_R32G32B32_TYPELESS	5	DXGI_FORMAT_R32G32B32_TYPELESS
DX_FORMAT_R32G32B32_FLOAT	6	DXGI_FORMAT_R32G32B32_FLOAT
DX_FORMAT_R32G32B32_UINT	7	DXGI_FORMAT_R32G32B32_UINT
DX_FORMAT_R32G32B32_SINT	8	DXGI_FORMAT_R32G32B32_SINT
DX_FORMAT_R16G16B16A16_TYPELESS	9	DXGI_FORMAT_R16G16B16A16_TYPELESS
DX_FORMAT_R16G16B16A16_FLOAT	10	DXGI_FORMAT_R16G16B16A16_FLOAT
DX_FORMAT_R16G16B16A16_UNORM	11	DXGI_FORMAT_R16G16B16A16_UNORM
DX_FORMAT_R16G16B16A16_UINT	12	DXGI_FORMAT_R16G16B16A16_UINT
DX_FORMAT_R16G16B16A16_SNORM	13	DXGI_FORMAT_R16G16B16A16_SNORM
DX_FORMAT_R16G16B16A16_SINT	14	DXGI_FORMAT_R16G16B16A16_SINT
DX_FORMAT_R32G32_TYPELESS	15	DXGI_FORMAT_R32G32_TYPELESS
DX_FORMAT_R32G32_FLOAT	16	DXGI_FORMAT_R32G32_FLOAT
DX_FORMAT_R32G32_UINT	17	DXGI_FORMAT_R32G32_UINT
DX_FORMAT_R32G32_SINT	18	DXGI_FORMAT_R32G32_SINT
DX_FORMAT_R32G8X24_TYPELESS	19	DXGI_FORMAT_R32G8X24_TYPELESS
DX_FORMAT_D32_FLOAT_S8X24_UINT	20	DXGI_FORMAT_D32_FLOAT_S8X24_UINT
DX_FORMAT_R32_FLOAT_X8X24_TYPELESS	21	DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS
DX_FORMAT_X32_TYPELESS_G8X24_UINT	22	DXGI_FORMAT_X32_TYPELESS_G8X24_UINT
DX_FORMAT_R10G10B10A2_TYPELESS	23	DXGI_FORMAT_R10G10B10A2_TYPELESS
DX_FORMAT_R10G10B10A2_UNORM	24	DXGI_FORMAT_R10G10B10A2_UNORM

ID	Value	Match in DXGI_FORMAT
DX_FORMAT_R10G10B10A2_UINT	25	DXGI_FORMAT_R10G10B10A2_UINT
DX_FORMAT_R11G11B10_FLOAT	26	DXGI_FORMAT_R11G11B10_FLOAT
DX_FORMAT_R8G8B8A8_TYPELESS	27	DXGI_FORMAT_R8G8B8A8_TYPELESS
DX_FORMAT_R8G8B8A8_UNORM	28	DXGI_FORMAT_R8G8B8A8_UNORM
DX_FORMAT_R8G8B8A8_UNORM_SRGB	29	DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
DX_FORMAT_R8G8B8A8_UINT	30	DXGI_FORMAT_R8G8B8A8_UINT
DX_FORMAT_R8G8B8A8_SNORM	31	DXGI_FORMAT_R8G8B8A8_SNORM
DX_FORMAT_R8G8B8A8_SINT	32	DXGI_FORMAT_R8G8B8A8_SINT
DX_FORMAT_R16G16_TYPELESS	33	DXGI_FORMAT_R16G16_TYPELESS
DX_FORMAT_R16G16_FLOAT	34	DXGI_FORMAT_R16G16_FLOAT
DX_FORMAT_R16G16_UNORM	35	DXGI_FORMAT_R16G16_UNORM
DX_FORMAT_R16G16_UINT	36	DXGI_FORMAT_R16G16_UINT
DX_FORMAT_R16G16_SNORM	37	DXGI_FORMAT_R16G16_SNORM
DX_FORMAT_R16G16_SINT	38	DXGI_FORMAT_R16G16_SINT
DX_FORMAT_R32_TYPELESS	39	DXGI_FORMAT_R32_TYPELESS
DX_FORMAT_D32_FLOAT	40	DXGI_FORMAT_D32_FLOAT
DX_FORMAT_R32_FLOAT	41	DXGI_FORMAT_R32_FLOAT
DX_FORMAT_R32_UINT	42	DXGI_FORMAT_R32_UINT
DX_FORMAT_R32_SINT	43	DXGI_FORMAT_R32_SINT
DX_FORMAT_R24G8_TYPELESS	44	DXGI_FORMAT_R24G8_TYPELESS
DX_FORMAT_D24_UNORM_S8_UINT	45	DXGI_FORMAT_D24_UNORM_S8_UINT
DX_FORMAT_R24_UNORM_X8_TYPELESS	46	DXGI_FORMAT_R24_UNORM_X8_TYPELESS
DX_FORMAT_X24_TYPELESS_G8_UINT	47	DXGI_FORMAT_X24_TYPELESS_G8_UINT
DX_FORMAT_R8G8_TYPELESS	48	DXGI_FORMAT_R8G8_TYPELESS
DX_FORMAT_R8G8_UNORM	49	DXGI_FORMAT_R8G8_UNORM
DX_FORMAT_R8G8_UINT	50	DXGI_FORMAT_R8G8_UINT
DX_FORMAT_R8G8_SNORM	51	DXGI_FORMAT_R8G8_SNORM
DX_FORMAT_R8G8_SINT	52	DXGI_FORMAT_R8G8_SINT
DX_FORMAT_R16_TYPELESS	53	DXGI_FORMAT_R16_TYPELESS

ID	Value	Match in DXGI_FORMAT
DX_FORMAT_R16_FLOAT	54	DXGI_FORMAT_R16_FLOAT
DX_FORMAT_D16_UNORM	55	DXGI_FORMAT_D16_UNORM
DX_FORMAT_R16_UNORM	56	DXGI_FORMAT_R16_UNORM
DX_FORMAT_R16_UINT	57	DXGI_FORMAT_R16_UINT
DX_FORMAT_R16_SNORM	58	DXGI_FORMAT_R16_SNORM
DX_FORMAT_R16_SINT	59	DXGI_FORMAT_R16_SINT
DX_FORMAT_R8_TYPELESS	60	DXGI_FORMAT_R8_TYPELESS
DX_FORMAT_R8_UNORM	61	DXGI_FORMAT_R8_UNORM
DX_FORMAT_R8_UINT	62	DXGI_FORMAT_R8_UINT
DX_FORMAT_R8_SNORM	63	DXGI_FORMAT_R8_SNORM
DX_FORMAT_R8_SINT	64	DXGI_FORMAT_R8_SINT
DX_FORMAT_A8_UNORM	65	DXGI_FORMAT_A8_UNORM
DX_FORMAT_R1_UNORM	66	DXGI_FORMAT_R1_UNORM
DX_FORMAT_R9G9B9E5_SHAREDEXP	67	DXGI_FORMAT_R9G9B9E5_SHAREDEXP
DX_FORMAT_R8G8_B8G8_UNORM	68	DXGI_FORMAT_R8G8_B8G8_UNORM
DX_FORMAT_G8R8_G8B8_UNORM	69	DXGI_FORMAT_G8R8_G8B8_UNORM
DX_FORMAT_BC1_TYPELESS	70	DXGI_FORMAT_BC1_TYPELESS
DX_FORMAT_BC1_UNORM	71	DXGI_FORMAT_BC1_UNORM
DX_FORMAT_BC1_UNORM_SRGB	72	DXGI_FORMAT_BC1_UNORM_SRGB
DX_FORMAT_BC2_TYPELESS	73	DXGI_FORMAT_BC2_TYPELESS
DX_FORMAT_BC2_UNORM	74	DXGI_FORMAT_BC2_UNORM
DX_FORMAT_BC2_UNORM_SRGB	75	DXGI_FORMAT_BC2_UNORM_SRGB
DX_FORMAT_BC3_TYPELESS	76	DXGI_FORMAT_BC3_TYPELESS
DX_FORMAT_BC3_UNORM	77	DXGI_FORMAT_BC3_UNORM
DX_FORMAT_BC3_UNORM_SRGB	78	DXGI_FORMAT_BC3_UNORM_SRGB
DX_FORMAT_BC4_TYPELESS	79	DXGI_FORMAT_BC4_TYPELESS
DX_FORMAT_BC4_UNORM	80	DXGI_FORMAT_BC4_UNORM
DX_FORMAT_BC4_SNORM	81	DXGI_FORMAT_BC4_SNORM
DX_FORMAT_BC5_TYPELESS	82	DXGI_FORMAT_BC5_TYPELESS
DX_FORMAT_BC5_UNORM	83	DXGI_FORMAT_BC5_UNORM
DX_FORMAT_BC5_SNORM	84	DXGI_FORMAT_BC5_SNORM

ID	Value	Match in DXGI_FORMAT
DX_FORMAT_B5G6R5_UNORM	85	DXGI_FORMAT_B5G6R5_UNORM
DX_FORMAT_B5G5R5A1_UNORM	86	DXGI_FORMAT_B5G5R5A1_UNORM
DX_FORMAT_B8G8R8A8_UNORM	87	DXGI_FORMAT_B8G8R8A8_UNORM
DX_FORMAT_B8G8R8X8_UNORM	88	DXGI_FORMAT_B8G8R8X8_UNORM
DX_FORMAT_R10G10B10_XR_BIAS_A2_UNORM	89	DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM
DX_FORMAT_B8G8R8A8_TYPELESS	90	DXGI_FORMAT_B8G8R8A8_TYPELESS
DX_FORMAT_B8G8R8A8_UNORM_SRGB	91	DXGI_FORMAT_B8G8R8A8_UNORM_SRGB
DX_FORMAT_B8G8R8X8_TYPELESS	92	DXGI_FORMAT_B8G8R8X8_TYPELESS
DX_FORMAT_B8G8R8X8_UNORM_SRGB	93	DXGI_FORMAT_B8G8R8X8_UNORM_SRGB
DX_FORMAT_BC6H_TYPELESS	94	DXGI_FORMAT_BC6H_TYPELESS
DX_FORMAT_BC6H_UF16	95	DXGI_FORMAT_BC6H_UF16
DX_FORMAT_BC6H_SF16	96	DXGI_FORMAT_BC6H_SF16
DX_FORMAT_BC7_TYPELESS	97	DXGI_FORMAT_BC7_TYPELESS
DX_FORMAT_BC7_UNORM	98	DXGI_FORMAT_BC7_UNORM
DX_FORMAT_BC7_UNORM_SRGB	99	DXGI_FORMAT_BC7_UNORM_SRGB
DX_FORMAT_AYUV	100	DXGI_FORMAT_AYUV
DX_FORMAT_Y410	101	DXGI_FORMAT_Y410
DX_FORMAT_Y416	102	DXGI_FORMAT_Y416
DX_FORMAT_NV12	103	DXGI_FORMAT_NV12
DX_FORMAT_P010	104	DXGI_FORMAT_P010
DX_FORMAT_P016	105	DXGI_FORMAT_P016
DX_FORMAT_420_OPAQUE	106	DXGI_FORMAT_420_OPAQUE
DX_FORMAT_YUY2	107	DXGI_FORMAT_YUY2
DX_FORMAT_Y210	108	DXGI_FORMAT_Y210
DX_FORMAT_Y216	109	DXGI_FORMAT_Y216
DX_FORMAT_NV11	110	DXGI_FORMAT_NV11
DX_FORMAT_AI44	111	DXGI_FORMAT_AI44
DX_FORMAT_IA44	112	DXGI_FORMAT_IA44
DX_FORMAT_P8	113	DXGI_FORMAT_P8

ID	Value	Match in DXGI_FORMAT
DX_FORMAT_A8P8	114	DXGI_FORMAT_A8P8
DX_FORMAT_B4G4R4A4_UNORM	115	DXGI_FORMAT_B4G4R4A4_UNORM
DX_FORMAT_P208	130	DXGI_FORMAT_P208
DX_FORMAT_V208	131	DXGI_FORMAT_V208
DX_FORMAT_V408	132	DXGI_FORMAT_V408
DX_FORMAT_FORCE_UINT	0xffffffff	DXGI_FORMAT_FORCE_UINT

DXInputCreate

Creates shader inputs.

```
int  DXInputCreate(
   int    context,         // graphic context handle
   uint   input_size       // size of inputs in bytes
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

input_size

[in] Size of the parameter structure in bytes.

Return Value

The handle for shader inputs or INVALID_HANDLE in case of an error. To receive an error code, the GetLastError() function should be called.

A created handle that is no longer in use should be explicitly released by the DXRelease() function.

DXInputSet

Sets shader inputs.

```
bool  DXInputSet(
   int          input,      // graphic context handle
   const void&  data        // data for setting
   );
```

Parameters

input

[in] Handle of inputs for a shader obtained in DXInputCreate().

data

[in] Data for setting shader inputs.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

DXShaderCreate

Creates a shader of a specified type.

```
int  DXShaderCreate(
   int                   context,          // graphic context handle
   ENUM_DX_SHADER_TYPE   shader_type,      // shader type
   const string          source,           // shader source code
   const string          entry_point,      // entry point
   string&               compile_error     // string for receiving compiler messages
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

shader_type

[out] The value from the ENUM_DX_SHADER_TYPE enumeration.

source

[in] Shader source code in HLSL 5.

entry_point

[in] Entry point – function name in a source code.

compile_error

[in] String for receiving compilation errors.

Return Value

Handle for shader or INVALID_HANDLE in case of an error. To receive an error code, the GetLastError() function should be called.

Note

A created handle that is no longer in use should be explicitly released by the DXRelease() function.

ENUM_DX_SHADER_TYPE

ID	Value	Description
DX_SHADER_VERTEX	0	Vertex shader
DX_SHADER_GEOMETRY	1	Geometry shader
DX_SHADER_PIXEL	2	Pixel shader

DXShaderSetLayout

Sets vertex layout for the vertex shader.

```
bool  DXShaderSetLayout(
   int                      shader,       // shader handle
   const DXVertexLayout&    layout[]      //
   );
```

Parameters

shader

[in] Handle of a vertex shader created in DXShaderCreate().

layout[]

[in] Array of vertex fields description. The description is set by the DXVertexLayout structure:

```
struct DXVertexLayout
  {
   string          semantic_name;       // The HLSL semantic associated with this e
   uint            semantic_index;      // The semantic index for the element. A se
   ENUM_DX_FORMAT  format;              // The data type of the element data.
  };
```

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The layout should match the type of vertices in a specified vertex buffer. It should also match the input type of vertices used at the entry point in the vertex shader code.

The vertex buffer for a shader is set in DXBufferSet().

The DXVertexLayout structure is a version of the D3D11_INPUT_ELEMENT_DESC MSDN structure.

DXShaderInputsSet

Sets shader inputs.

```
bool  DXShaderInputsSet(
   int          shader,       // shader handle
   const int&   inputs[]      // array of input handles
   );
```

Parameters

shader

[in] Handle of a shader created in DXShaderCreate().

inputs[]

[in] Array of input handles created using DXInputCreate().

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The size of the input parameter should be equal to the number of cbuffer objects declared in the shader code.

DXShaderTexturesSet

Sets shader textures.

```
bool  DXShaderTexturesSet(
   int          shader,         // shader handle
   const  int&  textures[]      // array of structure handles
   );
```

Parameters

shader

[in] Handle of a shader created in DXShaderCreate().

textures[]

[in] Array of texture handles created using DXTextureCreate().

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The size of the texture array should be equal to the number of Texture2D objects declared in the shader code.

DXDraw

Renders the vertices of the vertex buffer set in DXBufferSet().

```
bool  DXDraw(
   int    context,                // graphic context handle
   uint   start=0,                // first vertex index
   uint   count=WHOLE_ARRAY       // number of vertices
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

start=0

[in] Index of the first vertex for rendering.

count=WHOLE_ARRAY

[in] Number of vertices to render.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

Shaders should be preliminarily set using DXShaderSet() for rendering vertices.

DXDrawIndexed

Renders graphic primitives described by the index buffer from DXBufferSet().

```
bool  DXDrawIndexed(
   int    context,               // graphic context handle
   uint   start=0,               // first primitive index
   uint   count=WHOLE_ARRAY      // number of primitives
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

start=0

[in] Index of the first primitive for rendering.

count=WHOLE_ARRAY

[in] Number of primitives for rendering.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The type of primitives described by the index buffer is set using DXPrimiveTopologySet().

The vertex buffer in DXBufferSet() should be preliminarily set to render primitives.

Also, shaders should be preliminarily set using DXShaderSet().

DXPrimiveTopologySet

Sets the type of primitives for rendering using DXDrawIndexed().

```
bool  DXPrimiveTopologySet(
   int                          context,                // graphic context handle
   ENUM_DX_PRIMITIVE_TOPOLOGY   primitive_topology      // primitive type
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

primitive_topology

[in] The value from the ENUM_DX_PRIMITIVE_TOPOLOGY enumeration.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

ENUM_DX_PRIMITIVE_TOPOLOGY

ID	Value	Match in D3D11_PRIMITIVE_TOPOLOGY
DX_PRIMITIVE_TOPOLOGY_POINTLIST	1	D3D11_PRIMITIVE_TOPOLOGY_POINTLIST
DX_PRIMITIVE_TOPOLOGY_LINELIST	2	D3D11_PRIMITIVE_TOPOLOGY_LINELIST
DX_PRIMITIVE_TOPOLOGY_LINESTRIP	3	D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP
DX_PRIMITIVE_TOPOLOGY_TRIANGLELIST	4	D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST
DX_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP	5	D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP
DX_PRIMITIVE_TOPOLOGY_LINELIST_ADJ	6	D3D11_PRIMITIVE_TOPOLOGY_LINELIST_ADJ
DX_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ	7	D3D11_PRIMITIVE_TOPOLOGY_LINESTRIP_ADJ
DX_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ	8	D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST_ADJ
DX_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ	9	D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP_ADJ

DXBufferSet

Sets a buffer for the current rendering.

```
bool  DXBufferSet(
   int    context,               // graphic context handle
   int    buffer,                // vertex or index buffer handle
   uint   start=0,               // initial index
   uint   count=WHOLE_ARRAY      // number of elements
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

buffer

[in] Handle of the vertex or index buffer created in DXBufferCreate().

start=0

[in] Index of the buffer first element. The data from the beginning of the buffer is used by default.

count=WHOLE_ARRAY

[in] Number of values to be used. The default is all buffer values.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

The DXBufferSet() function should be called to set vertex and index buffers for rendering using DXDraw().

DXShaderSet

Sets a shader for rendering.

```
bool  DXShaderSet(
   int   context,       // graphic context handle
   int   shader         // shader handle
   );
```

Parameters

context

[in] Handle for a graphic context created in DXContextCreate().

shader

[in] Handle of a shader created in DXShaderCreate().

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

Several types of shaders can simultaneously be used for rendering (vertex, geometry and pixel ones).

DXHandleType

Returns a handle type.

```
ENUM_DX_HANDLE_TYPE  DXHandleType(
   int   handle       // handle
   );
```

Parameters

handle

[in] Handle.

Return Value

The value from the ENUM_DX_HANDLE_TYPE enumeration

ENUM_DX_HANDLE_TYPE

ID	Value	Description
DX_HANDLE_INVALID	0	Invalid handle
DX_HANDLE_CONTEXT	1	Graphic context handle
DX_HANDLE_SHADER	2	Shader handle
DX_HANDLE_BUFFER	3	Vertex or index buffer handle
DX_HANDLE_INPUT	4	Handle for shader inputs
DX_HANDLE_TEXTURE	5	Texture handle

DXRelease

Releases a handle.

```
bool  DXRelease(
   int   handle        // handle
   );
```

Parameters

context

[in] Released handle.

Return Value

In case of successful execution, returns true, otherwise - false. To receive an error code, the GetLastError() function should be called.

Note

All created handles that are no longer in use should be explicitly released by the DXRelease() function.

MetaTrader module for integration with Python

MQL5 is designed for the development of high-performance trading applications in the financial markets and is unparalleled among other specialized languages used in the algorithmic trading. The syntax and speed of MQL5 programs are very close to C++, there is support for OpenCL and integration with MS Visual Studio. Statistics, fuzzy logic and ALGLIB libraries are available as well. MetaEditor development environment features native support for .NET libraries with "smart" functions import eliminating the need to develop special wrappers. Third-party C++ DLLs can also be used. C++ source code files (CPP and H) can be edited and compiled into DLL directly from the editor. Microsoft Visual Studio installed on user's PC can be used for that.

Python is a modern high-level programming language for developing scripts and applications. It contains multiple libraries for machine learning, process automation, as well as data analysis and visualization.

MetaTrader package for Python is designed for convenient and fast obtaining of exchange data via interprocessor communication directly from the MetaTrader 5 terminal. The data received this way can be further used for statistical calculations and machine learning.

Installing the package from the command line:

```
pip install MetaTrader5
```

Updating the package from the command line:

```
pip install --upgrade MetaTrader5
```

Functions for integrating MetaTrader 5 and Python

Function	Action
initialize	Establish a connection with the MetaTrader 5 terminal
login	Connect to a trading account using specified parameters
shutdown	Close the previously established connection to the MetaTrader 5 terminal
version	Return the MetaTrader 5 terminal version
last_error	Return data on the last error
account_info	Get info on the current trading account
terminal_Info	Get status and parameters of the connected MetaTrader 5 terminal
symbols_total	Get the number of all financial instruments in the MetaTrader 5 terminal
symbols_get	Get all financial instruments from the MetaTrader 5 terminal
symbol_info	Get data on the specified financial instrument
symbol_info_tick	Get the last tick for the specified financial instrument
symbol_select	Select a symbol in the MarketWatch window or remove a symbol from the window

Function	Action
market_book_add	Subscribes the MetaTrader 5 terminal to the Market Depth change events for a specified symbol
market_book_get	Returns a tuple from BookInfo featuring Market Depth entries for the specified symbol
market_book_release	Cancels subscription of the MetaTrader 5 terminal to the Market Depth change events for a specified symbol
copy_rates_from	Get bars from the MetaTrader 5 terminal starting from the specified date
copy_rates_from_pos	Get bars from the MetaTrader 5 terminal starting from the specified index
copyrates_range	Get bars in the specified date range from the MetaTrader 5 terminal
copy_ticks_from	Get ticks from the MetaTrader 5 terminal starting from the specified date
copy_ticks_range	Get ticks for the specified date range from the MetaTrader 5 terminal
orders_total	Get the number of active orders.
orders_get	Get active orders with the ability to filter by symbol or ticket
order_calc_margin	Return margin in the account currency to perform a specified trading operation
order_calc_profit	Return profit in the account currency for a specified trading operation
order_check	Check funds sufficiency for performing a required trading operation
order_send	Send a request to perform a trading operation.
positions_total	Get the number of open positions
positions_get	Get open positions with the ability to filter by symbol or ticket
history_orders_total	Get the number of orders in trading history within the specified interval
history_orders_get	Get orders from trading history with the ability to filter by ticket or position
history_deals_total	Get the number of deals in trading history within the specified interval
history_deals_get	Get deals from trading history with the ability to filter by ticket or position

Example of connecting Python to MetaTrader 5

1. Download the latest version of Python 3.8 from https://www.python.org/downloads/windows
2. When installing Python, check "Add Python 3.8 to PATH%" to be able to run Python scripts from the command line.

3. Install the MetaTrader 5 module from the command line

```
pip install MetaTrader5
```

4. Add matplotlib and pandas packages

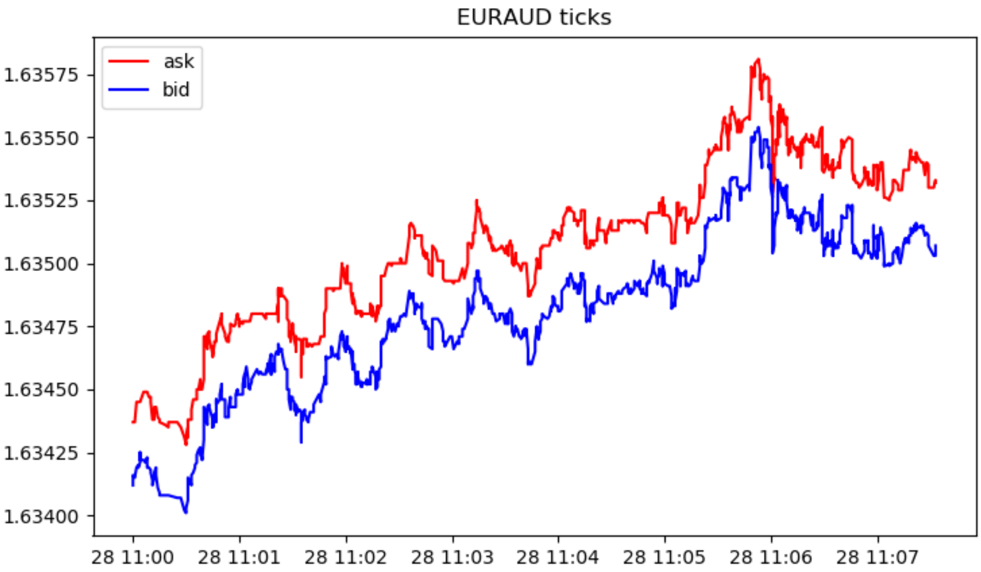
```
pip install matplotlib
pip install pandas
```

5. Launch the test script

```
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
import MetaTrader5 as mt5

# connect to MetaTrader 5
if not mt5.initialize():
    print("initialize() failed")
    mt5.shutdown()

# request connection status and parameters
print(mt5.terminal_info())
# get data on MetaTrader 5 version
print(mt5.version())

# request 1000 ticks from EURAUD
euraud_ticks = mt5.copy_ticks_from("EURAUD", datetime(2020,1,28,13), 1000, mt5.COPY_TI
# request ticks from AUDUSD within 2019.04.01 13:00 - 2019.04.02 13:00
audusd_ticks = mt5.copy_ticks_range("AUDUSD", datetime(2020,1,27,13), datetime(2020,1,

# get bars from different symbols in a number of ways
eurusd_rates = mt5.copy_rates_from("EURUSD", mt5.TIMEFRAME_M1, datetime(2020,1,28,13),
eurgbp_rates = mt5.copy_rates_from_pos("EURGBP", mt5.TIMEFRAME_M1, 0, 1000)
eurcad_rates = mt5.copy_rates_range("EURCAD", mt5.TIMEFRAME_M1, datetime(2020,1,27,13)

# shut down connection to MetaTrader 5
mt5.shutdown()

#DATA
print('euraud_ticks(', len(euraud_ticks), ')')
for val in euraud_ticks[:10]: print(val)

print('audusd_ticks(', len(audusd_ticks), ')')
for val in audusd_ticks[:10]: print(val)

print('eurusd_rates(', len(eurusd_rates), ')')
for val in eurusd_rates[:10]: print(val)

print('eurgbp_rates(', len(eurgbp_rates), ')')
```

```
for val in eurgbp_rates[:10]: print(val)

print('eurcad_rates(', len(eurcad_rates), ')')
for val in eurcad_rates[:10]: print(val)

#PLOT
# create DataFrame out of the obtained data
ticks_frame = pd.DataFrame(euraud_ticks)
# convert time in seconds into the datetime format
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')
# display ticks on the chart
plt.plot(ticks_frame['time'], ticks_frame['ask'], 'r-', label='ask')
plt.plot(ticks_frame['time'], ticks_frame['bid'], 'b-', label='bid')

# display the legends
plt.legend(loc='upper left')

# add the header
plt.title('EURAUD ticks')

# display the chart
plt.show()
```

6. Get data and chart



```
[2, 'MetaQuotes-Demo', '16167573']
[500, 2325, '19 Feb 2020']

euraud_ticks( 1000 )
(1580209200, 1.63412, 1.63437, 0., 0, 1580209200067, 130, 0.)
(1580209200, 1.63416, 1.63437, 0., 0, 1580209200785, 130, 0.)
(1580209201, 1.63415, 1.63437, 0., 0, 1580209201980, 130, 0.)
(1580209202, 1.63419, 1.63445, 0., 0, 1580209202192, 134, 0.)
```

```
(1580209203, 1.6342, 1.63445, 0., 0, 1580209203004, 130, 0.)
(1580209203, 1.63419, 1.63445, 0., 0, 1580209203487, 130, 0.)
(1580209203, 1.6342, 1.63445, 0., 0, 1580209203694, 130, 0.)
(1580209203, 1.63419, 1.63445, 0., 0, 1580209203990, 130, 0.)
(1580209204, 1.63421, 1.63445, 0., 0, 1580209204194, 130, 0.)
(1580209204, 1.63425, 1.63445, 0., 0, 1580209204392, 130, 0.)
audusd_ticks( 40449 )
(1580122800, 0.67858, 0.67868, 0., 0, 1580122800244, 130, 0.)
(1580122800, 0.67858, 0.67867, 0., 0, 1580122800429, 4, 0.)
(1580122800, 0.67858, 0.67865, 0., 0, 1580122800817, 4, 0.)
(1580122801, 0.67858, 0.67866, 0., 0, 1580122801618, 4, 0.)
(1580122802, 0.67858, 0.67865, 0., 0, 1580122802928, 4, 0.)
(1580122809, 0.67855, 0.67865, 0., 0, 1580122809526, 130, 0.)
(1580122809, 0.67855, 0.67864, 0., 0, 1580122809699, 4, 0.)
(1580122813, 0.67855, 0.67863, 0., 0, 1580122813576, 4, 0.)
(1580122815, 0.67856, 0.67863, 0., 0, 1580122815190, 130, 0.)
(1580122815, 0.67855, 0.67863, 0., 0, 1580122815479, 130, 0.)
eurusd_rates( 1000 )
(1580149260, 1.10132, 1.10151, 1.10131, 1.10149, 44, 1, 0)
(1580149320, 1.10149, 1.10161, 1.10143, 1.10154, 42, 1, 0)
(1580149380, 1.10154, 1.10176, 1.10154, 1.10174, 40, 2, 0)
(1580149440, 1.10174, 1.10189, 1.10168, 1.10187, 47, 1, 0)
(1580149500, 1.10185, 1.10191, 1.1018, 1.10182, 53, 1, 0)
(1580149560, 1.10182, 1.10184, 1.10176, 1.10183, 25, 3, 0)
(1580149620, 1.10183, 1.10187, 1.10177, 1.10187, 49, 2, 0)
(1580149680, 1.10187, 1.1019, 1.1018, 1.10187, 53, 1, 0)
(1580149740, 1.10187, 1.10202, 1.10187, 1.10198, 28, 2, 0)
(1580149800, 1.10198, 1.10198, 1.10183, 1.10188, 39, 2, 0)
eurgbp_rates( 1000 )
(1582236360, 0.83767, 0.83767, 0.83764, 0.83765, 23, 9, 0)
(1582236420, 0.83765, 0.83765, 0.83764, 0.83765, 15, 8, 0)
(1582236480, 0.83765, 0.83766, 0.83762, 0.83765, 19, 7, 0)
(1582236540, 0.83765, 0.83768, 0.83758, 0.83763, 39, 6, 0)
(1582236600, 0.83763, 0.83768, 0.83763, 0.83767, 21, 6, 0)
(1582236660, 0.83767, 0.83775, 0.83765, 0.83769, 63, 5, 0)
(1582236720, 0.83769, 0.8377, 0.83758, 0.83764, 40, 7, 0)
(1582236780, 0.83766, 0.83769, 0.8376, 0.83766, 37, 6, 0)
(1582236840, 0.83766, 0.83772, 0.83763, 0.83772, 22, 6, 0)
(1582236900, 0.83772, 0.83773, 0.83768, 0.8377, 36, 5, 0)
eurcad_rates( 1441 )
(1580122800, 1.45321, 1.45329, 1.4526, 1.4528, 146, 15, 0)
(1580122860, 1.4528, 1.45315, 1.45274, 1.45301, 93, 15, 0)
(1580122920, 1.453, 1.45304, 1.45264, 1.45264, 82, 15, 0)
(1580122980, 1.45263, 1.45279, 1.45231, 1.45277, 109, 15, 0)
(1580123040, 1.45275, 1.4528, 1.45259, 1.45271, 53, 14, 0)
(1580123100, 1.45273, 1.45285, 1.45269, 1.4528, 62, 16, 0)
(1580123160, 1.4528, 1.45284, 1.45267, 1.45282, 64, 14, 0)
(1580123220, 1.45282, 1.45299, 1.45261, 1.45272, 48, 14, 0)
(1580123280, 1.45272, 1.45275, 1.45255, 1.45275, 74, 14, 0)
```

```
(1580123340, 1.45275, 1.4528, 1.4526, 1.4528, 94, 13, 0)
```

initialize

Establish a connection with the MetaTrader 5 terminal. There are three call options.

Call without parameters. The terminal for connection is found automatically.

```
initialize()
```

Call specifying the path to the MetaTrader 5 terminal we want to connect to.

```
initialize(
   path                      // path to the MetaTrader 5 terminal EXE file
   )
```

Call specifying the trading account path and parameters.

```
initialize(
   path,                     // path to the MetaTrader 5 terminal EXE file
   login=LOGIN,              // account number
   password="PASSWORD",      // password
   server="SERVER",          // server name as it is specified in the terminal
   timeout=TIMEOUT,          // timeout
   portable=False            // portable mode
   )
```

Parameters

path

[in] Path to the metatrader.exe or metatrader64.exe file. Optional unnamed parameter. It is indicated first without a parameter name. If the path is not specified, the module attempts to find the executable file on its own.

login=LOGIN

[in] Trading account number. Optional named parameter. If not specified, the last trading account is used.

password="PASSWORD"

[in] Trading account password. Optional named parameter. If the password is not set, the password for a specified trading account saved in the terminal database is applied automatically.

server="SERVER"

[in] Trade server name. Optional named parameter. If the server is not set, the server for a specified trading account saved in the terminal database is applied automatically.

timeout=TIMEOUT

[in] Connection timeout in milliseconds. Optional named parameter. If not specified, the value of 60 000 (60 seconds) is applied.

portable=False

[in] Flag of the terminal launch in portable mode. Optional named parameter. If not specified, the value of False is used.

Return Value

Returns True in case of successful connection to the MetaTrader 5 terminal, otherwise - False.

Note

If required, the MetaTrader 5 terminal is launched to establish connection when executing the initialize() call.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish MetaTrader 5 connection to a specified trading account
if not mt5.initialize(login=25115284, server="MetaQuotes-Demo",password="4zatlbqx"):
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# display data on connection status, server name and trading account
print(mt5.terminal_info())
# display data on MetaTrader 5 version
print(mt5.version())

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

shutdown, terminal_info, version

login

Connect to a trading account using specified parameters.

```
login(
   login,                    // account number
   password="PASSWORD",      // password
   server="SERVER",          // server name as it is specified in the terminal
   timeout=TIMEOUT           // timeout
   )
```

Parameters

login

[in] Trading account number. Required unnamed parameter.

password

[in] Trading account password. Optional named parameter. If the password is not set, the password saved in the terminal database is applied automatically.

server

[in] Trade server name. Optional named parameter. If no server is set, the last used server is applied automatically.

timeout=TIMEOUT

[in] Connection timeout in milliseconds. Optional named parameter. If not specified, the value of 60 000 (60 seconds) is applied. If the connection is not established within the specified time, the call is forcibly terminated and the exception is generated.

Return Value

True in case of a successful connection to the trade account, otherwise – False.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# display data on MetaTrader 5 version
print(mt5.version())
# connect to the trade account without specifying a password and a server
account=17221085
authorized=mt5.login(account)  # the terminal database password is applied if connecti
if authorized:
    print("connected to account #{}".format(account))
else:
    print("failed to connect at account #{}, error code: {}".format(account, mt5.last_

# now connect to another trading account specifying the password
account=25115284
authorized=mt5.login(account, password="gqrtz0lbdm")
if authorized:
    # display trading account data 'as is'
    print(mt5.account_info())
    # display trading account data in the form of a list
    print("Show account_info()._asdict():")
    account_info_dict = mt5.account_info()._asdict()
    for prop in account_info_dict:
        print("  {}={}".format(prop, account_info_dict[prop]))
else:
    print("failed to connect at account #{}, error code: {}".format(account, mt5.last_

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
[500, 2367, '23 Mar 2020']

connected to account #17221085

connected to account #25115284
AccountInfo(login=25115284, trade_mode=0, leverage=100, limit_orders=200, margin_so_m
account properties:
   login=25115284
   trade_mode=0
   leverage=100
   limit_orders=200
   margin_so_mode=0
   trade_allowed=True
   trade_expert=True
   margin_mode=2
   currency_digits=2
   fifo_close=False
   balance=99588.33
   credit=0.0
   profit=-45.23
   equity=99543.1
   margin=54.37
   margin_free=99488.73
   margin_level=183084.6054809638
   margin_so_call=50.0
   margin_so_so=30.0
```

```
  margin_initial=0.0
  margin_maintenance=0.0
  assets=0.0
  liabilities=0.0
  commission_blocked=0.0
  name=James Smith
  server=MetaQuotes-Demo
  currency=USD
  company=MetaQuotes Software Corp.
```

See also

initialize, shutdown

shutdown

Close the previously established connection to the MetaTrader 5 terminal.

```
shutdown()
```

Return Value

None.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed")
    quit()

# display data on connection status, server name and trading account
print(mt5.terminal_info())
# display data on MetaTrader 5 version
print(mt5.version())

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

initialize, login_py, terminal_info, version

version

Return the MetaTrader 5 terminal version.

```
version()
```

Return Value

Return the MetaTrader 5 terminal version, build and release date. Return None in case of an error. The info on the error can be obtained using last_error().

Note

The version() function returns the terminal version, build and release date as a tuple of three values:

Type	Description	Sample value
integer	MetaTrader 5 terminal version	500
integer	Build	2007
string	Build release date	'25 Feb 2019'

Example:

```
import MetaTrader5 as mt5
import pandas as pd
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# display data on MetaTrader 5 version
print(mt5.version())

# display data on connection status, server name and trading account 'as is'
print(mt5.terminal_info())
print()

# get properties in the form of a dictionary
terminal_info_dict=mt5.terminal_info()._asdict()
# convert the dictionary into DataFrame and print
df=pd.DataFrame(list(terminal_info_dict.items()),columns=['property','value'])
print("terminal_info() as dataframe:")
print(df[:-1])
```

```
# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
[500, 2367, '23 Mar 2020']
TerminalInfo(community_account=True, community_connection=True, connected=True, dlls_a

terminal_info() as dataframe:
                 property                          value
0        community_account                          True
1     community_connection                          True
2                connected                          True
3             dlls_allowed                         False
4            trade_allowed                         False
5        tradeapi_disabled                         False
6            email_enabled                         False
7              ftp_enabled                         False
8    notifications_enabled                         False
9                     mqid                         False
10                   build                          2367
11                 maxbars                          5000
12                codepage                          1251
13               ping_last                         77881
14       community_balance                       707.107
15          retransmission                             0
16                 company     MetaQuotes Software Corp.
17                    name                  MetaTrader 5
18                language                       Russian
19                    path  E:\ProgramFiles\MetaTrader 5
20               data_path  E:\ProgramFiles\MetaTrader 5
```

See also

initialize, shutdown, terminal_info

last_error

Return data on the last error.

```
last_error()
```

Return Value

Return the last error code and description as a tuple.

Note

last_error() allows obtaining an error code in case of a failed execution of a MetaTrader 5 library function. It is similar to GetLastError(). However, it applies its own error codes. Possible values:

Constant	Value	Description
RES_S_OK	1	generic success
RES_E_FAIL	-1	generic fail
RES_E_INVALID_PARAMS	-2	invalid arguments/parameters
RES_E_NO_MEMORY	-3	no memory condition
RES_E_NOT_FOUND	-4	no history
RES_E_INVALID_VERSION	-5	invalid version
RES_E_AUTH_FAILED	-6	authorization failed
RES_E_UNSUPPORTED	-7	unsupported method
RES_E_AUTO_TRADING_DISABLED	-8	auto-trading disabled
RES_E_INTERNAL_FAIL	-10000	internal IPC general error
RES_E_INTERNAL_FAIL_SEND	-10001	internal IPC send failed
RES_E_INTERNAL_FAIL_RECEIVE	-10002	internal IPC recv failed
RES_E_INTERNAL_FAIL_INIT	-10003	internal IPC initialization fail
RES_E_INTERNAL_FAIL_CONNECT	-10003	internal IPC no ipc
RES_E_INTERNAL_FAIL_TIMEOUT	-10005	internal timeout

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()
```

```
# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

version, GetLastError

account_info

Get info on the current trading account.

```
account_info()
```

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function returns all data that can be obtained using AccountInfoInteger, AccountInfoDouble and AccountInfoString in one call.

Example:

```
import MetaTrader5 as mt5
import pandas as pd
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# connect to the trade account specifying a password and a server
authorized=mt5.login(25115284, password="gqz0343lbdm")
if authorized:
    account_info=mt5.account_info()
    if account_info!=None:
        # display trading account data 'as is'
        print(account_info)
        # display trading account data in the form of a dictionary
        print("Show account_info()._asdict():")
        account_info_dict = mt5.account_info()._asdict()
        for prop in account_info_dict:
            print("  {}={}".format(prop, account_info_dict[prop]))
        print()

        # convert the dictionary into DataFrame and print
        df=pd.DataFrame(list(account_info_dict.items()),columns=['property','value'])
        print("account_info() as dataframe:")
        print(df)
else:
    print("failed to connect to trade account 25115284 with password=gqz0343lbdm, erro

# shut down connection to the MetaTrader 5 terminal
```

```
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
AccountInfo(login=25115284, trade_mode=0, leverage=100, limit_orders=200, margin_so_m
Show account_info()._asdict():
  login=25115284
  trade_mode=0
  leverage=100
  limit_orders=200
  margin_so_mode=0
  trade_allowed=True
  trade_expert=True
  margin_mode=2
  currency_digits=2
  fifo_close=False
  balance=99511.4
  credit=0.0
  profit=41.82
  equity=99553.22
  margin=98.18
  margin_free=99455.04
  margin_level=101398.67590140559
  margin_so_call=50.0
  margin_so_so=30.0
  margin_initial=0.0
  margin_maintenance=0.0
  assets=0.0
  liabilities=0.0
  commission_blocked=0.0
  server=MetaQuotes-Demo
  currency=USD
  company=MetaQuotes Software Corp.

account_info() as dataframe
              property                      value
0                login                   25115284
1           trade_mode                          0
2             leverage                        100
3         limit_orders                        200
4       margin_so_mode                          0
5        trade_allowed                       True
6         trade_expert                       True
7          margin_mode                          2
8      currency_digits                          2
9           fifo_close                      False
10             balance                    99588.3
```

```
11              credit                          0
12              profit                     -45.13
13              equity                    99543.2
14              margin                      54.37
15         margin_free                    99488.8
16        margin_level                     183085
17      margin_so_call                         50
18        margin_so_so                         30
19      margin_initial                          0
20  margin_maintenance                          0
21              assets                          0
22         liabilities                          0
23  commission_blocked                          0
24                name                James Smith
25              server            MetaQuotes-Demo
26            currency                        USD
27             company  MetaQuotes Software Corp.
```

See also

initialize, shutdown, login

terminal_info

Get the connected MetaTrader 5 client terminal status and settings.

```
terminal_info()
```

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function returns all data that can be obtained using TerminalInfoInteger, TerminalInfoDouble and TerminalInfoDouble in one call.

Example:

```
import MetaTrader5 as mt5
import pandas as pd
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# display data on MetaTrader 5 version
print(mt5.version())
# display info on the terminal settings and status
terminal_info=mt5.terminal_info()
if terminal_info!=None:
    # display the terminal data 'as is'
    print(terminal_info)
    # display data in the form of a list
    print("Show terminal_info()._asdict():")
    terminal_info_dict = mt5.terminal_info()._asdict()
    for prop in terminal_info_dict:
        print("  {}={}".format(prop, terminal_info_dict[prop]))
    print()
   # convert the dictionary into DataFrame and print
    df=pd.DataFrame(list(terminal_info_dict.items()),columns=['property','value'])
    print("terminal_info() as dataframe:")
    print(df)

# shut down connection to the MetaTrader 5 terminal
```

```
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
[500, 2366, '20 Mar 2020']
TerminalInfo(community_account=True, community_connection=True, connected=True,....
Show terminal_info()._asdict():
  community_account=True
  community_connection=True
  connected=True
  dlls_allowed=False
  trade_allowed=False
  tradeapi_disabled=False
  email_enabled=False
  ftp_enabled=False
  notifications_enabled=False
  mqid=False
  build=2366
  maxbars=5000
  codepage=1251
  ping_last=77850
  community_balance=707.10668201585
  retransmission=0.0
  company=MetaQuotes Software Corp.
  name=MetaTrader 5
  language=Russian
  path=E:\ProgramFiles\MetaTrader 5
  data_path=E:\ProgramFiles\MetaTrader 5
  commondata_path=C:\Users\Rosh\AppData\Roaming\MetaQuotes\Terminal\Common

terminal_info() as dataframe:
                 property                      value
0       community_account                       True
1    community_connection                       True
2               connected                       True
3            dlls_allowed                      False
4           trade_allowed                      False
5       tradeapi_disabled                      False
6           email_enabled                      False
7             ftp_enabled                      False
8   notifications_enabled                      False
9                    mqid                      False
10                  build                       2367
11                maxbars                       5000
12               codepage                       1251
13              ping_last                      80953
14      community_balance                    707.107
```

```
15         retransmission                   0.063593
16                company  MetaQuotes Software Corp.
17                   name               MetaTrader 5
18               language                    Russian
```

See also

initialize, shutdown, version

symbols_total

Get the number of all financial instruments in the MetaTrader 5 terminal.

```
symbols_total()
```

Return Value

Integer value.

Note

The function is similar to SymbolsTotal(). However, it returns the number of all symbols including custom ones and the ones disabled in MarketWatch.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get the number of financial instruments
symbols=mt5.symbols_total()
if symbols>0:
    print("Total symbols =",symbols)
else:
    print("symbols not found")

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

symbols_get, symbol_select, symbol_info

symbols_get

Get all financial instruments from the MetaTrader 5 terminal.

```
symbols_get(
   group="GROUP"      // symbol selection filter
)
```

`group="GROUP"`

[in] The filter for arranging a group of necessary symbols. Optional parameter. If the group is specified, the function returns only symbols meeting a specified criteria.

Return Value

Return symbols in the form of a tuple. Return None in case of an error. The info on the error can be obtained using last_error().

Note

The *group* parameter allows sorting out symbols by name. '*' can be used at the beginning and the end of a string.

The *group* parameter can be used as a named or an unnamed one. Both options work the same way. The named option (group="GROUP") makes the code easier to read.

The *group* parameter may contain several comma separated conditions. A condition can be set as a mask using '*'. The logical negation symbol '!' can be used for an exclusion. All conditions are applied sequentially, which means conditions of including to a group should be specified first followed by an exclusion condition. For example, *group*="*, !EUR" means that all symbols should be selected first and the ones containing "EUR" in their names should be excluded afterwards.

Unlike symbol_info(), the symbols_get() function returns data on all requested symbols within a single call.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get all symbols
symbols=mt5.symbols_get()
print('Symbols: ', len(symbols))
count=0
# display the first five ones
for s in symbols:
    count+=1
    print("{}. {}".format(count,s.name))
    if count==5: break
print()

# get symbols containing RU in their names
ru_symbols=mt5.symbols_get("*RU*")
print('len(*RU*): ', len(ru_symbols))
for s in ru_symbols:
    print(s.name)
print()

# get symbols whose names do not contain USD, EUR, JPY and GBP
group_symbols=mt5.symbols_get(group="*,!*USD*,!*EUR*,!*JPY*,!*GBP*")
print('len(*,!*USD*,!*EUR*,!*JPY*,!*GBP*):', len(group_symbols))
for s in group_symbols:
    print(s.name,":",s)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
Symbols:  84
1. EURUSD
2. GBPUSD
3. USDCHF
4. USDJPY
5. USDCNH

len(*RU*):  8
EURUSD
USDRUB
USDRUR
EURRUR
EURRUB
FORTS.RUB.M5
EURUSD_T20
EURUSD4

len(*,!*USD*,!*EUR*,!*JPY*,!*GBP*):  13
AUDCAD : SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_dea
AUDCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
AUDNZD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
CADCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
NZDCAD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
NZDCHF : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
NZDSGD : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
CADMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
CHFMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
NZDMXN : SymbolInfo(custom=False, chart_mode=0, select=False, visible=False, session_
FORTS.RTS.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
FORTS.RUB.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
FOREX.CHF.M5 : SymbolInfo(custom=True, chart_mode=0, select=False, visible=False, sess
```

See also

symbols_total, symbol_select, symbol_info

symbol_info

Get data on the specified financial instrument.

```
symbol_info(
   symbol       // financial instrument name
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function returns all data that can be obtained using SymbolInfoInteger, SymbolInfoDouble and SymbolInfoString in one call.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# attempt to enable the display of the EURJPY symbol in MarketWatch
selected=mt5.symbol_select("EURJPY",True)
if not selected:
    print("Failed to select EURJPY")
    mt5.shutdown()
    quit()

# display EURJPY symbol properties
symbol_info=mt5.symbol_info("EURJPY")
if symbol_info!=None:
    # display the terminal data 'as is'
    print(symbol_info)
    print("EURJPY: spread =",symbol_info.spread,"  digits =",symbol_info.digits)
    # display symbol properties as a list
    print("Show symbol_info(\"EURJPY\")._asdict():")
    symbol_info_dict = mt5.symbol_info("EURJPY")._asdict()
    for prop in symbol_info_dict:
        print("  {}={}".format(prop, symbol_info_dict[prop]))
```

```
# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, ses
EURJPY: spread = 17   digits = 3
Show symbol_info()._asdict():
  custom=False
  chart_mode=0
  select=True
  visible=True
  session_deals=0
  session_buy_orders=0
  session_sell_orders=0
  volume=0
  volumehigh=0
  volumelow=0
  time=1585069682
  digits=3
  spread=17
  spread_float=True
  ticks_bookdepth=10
  trade_calc_mode=0
  trade_mode=4
  start_time=0
  expiration_time=0
  trade_stops_level=0
  trade_freeze_level=0
  trade_exemode=1
  swap_mode=1
  swap_rollover3days=3
  margin_hedged_use_leg=False
  expiration_mode=7
  filling_mode=1
  order_mode=127
  order_gtc_mode=0
  option_mode=0
  option_right=0
  bid=120.024
  bidhigh=120.506
  bidlow=118.798
  ask=120.041
  askhigh=120.526
  asklow=118.828
  last=0.0
```

```
lasthigh=0.0
lastlow=0.0
volume_real=0.0
volumehigh_real=0.0
volumelow_real=0.0
option_strike=0.0
point=0.001
trade_tick_value=0.8977708350166538
trade_tick_value_profit=0.8977708350166538
trade_tick_value_loss=0.8978272580355541
trade_tick_size=0.001
trade_contract_size=100000.0
trade_accrued_interest=0.0
trade_face_value=0.0
trade_liquidity_rate=0.0
volume_min=0.01
volume_max=500.0
volume_step=0.01
volume_limit=0.0
swap_long=-0.2
swap_short=-1.2
margin_initial=0.0
margin_maintenance=0.0
session_volume=0.0
session_turnover=0.0
session_interest=0.0
session_buy_orders_volume=0.0
session_sell_orders_volume=0.0
session_open=0.0
session_close=0.0
session_aw=0.0
session_price_settlement=0.0
session_price_limit_min=0.0
session_price_limit_max=0.0
margin_hedged=100000.0
price_change=0.0
price_volatility=0.0
price_theoretical=0.0
price_greeks_delta=0.0
price_greeks_theta=0.0
price_greeks_gamma=0.0
price_greeks_vega=0.0
price_greeks_rho=0.0
price_greeks_omega=0.0
price_sensitivity=0.0
basis=
category=
currency_base=EUR
currency_profit=JPY
```

```
currency_margin=EUR
bank=
description=Euro vs Japanese Yen
exchange=
formula=
isin=
name=EURJPY
page=http://www.google.com/finance?q=EURJPY
path=Forex\EURJPY
```

See also

account_info, terminal_info

symbol_info_tick

Get the last tick for the specified financial instrument.

```
symbol_info_tick(
   symbol      // financial instrument name
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

Return Value

Return info in the form of a tuple. Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function is similar to SymbolInfoTick.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# attempt to enable the display of the GBPUSD in MarketWatch
selected=mt5.symbol_select("GBPUSD",True)
if not selected:
    print("Failed to select GBPUSD")
    mt5.shutdown()
    quit()

# display the last GBPUSD tick
lasttick=mt5.symbol_info_tick("GBPUSD")
print(lasttick)
# display tick field values in the form of a list
print("Show symbol_info_tick(\"GBPUSD\")._asdict():")
symbol_info_tick_dict = mt5.symbol_info_tick("GBPUSD")._asdict()
for prop in symbol_info_tick_dict:
    print("  {}={}".format(prop, symbol_info_tick_dict[prop]))

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

```
Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
Tick(time=1585070338, bid=1.17264, ask=1.17279, last=0.0, volume=0, time_msc=158507033
Show symbol_info_tick._asdict():
  time=1585070338
  bid=1.17264
  ask=1.17279
  last=0.0
  volume=0
  time_msc=1585070338728
  flags=2
  volume_real=0.0
```

See also

symbol_info, symbol_info

symbol_select

Select a symbol in the MarketWatch window or remove a symbol from the window.

```
symbol_select(
   symbol,       // financial instrument name
   enable=None   // enable or disable
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

enable

[in] Switch. Optional unnamed parameter. If 'false', a symbol should be removed from the MarketWatch window. Otherwise, it should be selected in the MarketWatch window. A symbol cannot be removed if open charts with this symbol are currently present or positions are opened on it.

Return Value

True if successful, otherwise - False.

Note

The function is similar to SymbolSelect.

Example:

```
import MetaTrader5 as mt5
import pandas as pd
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# establish connection to the MetaTrader 5 terminal
if not mt5.initialize(login=25115284, server="MetaQuotes-Demo",password="4zatlbqx"):
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# attempt to enable the display of the EURCAD in MarketWatch
selected=mt5.symbol_select("EURCAD",True)
if not selected:
    print("Failed to select EURCAD, error code =",mt5.last_error())
else:
    symbol_info=mt5.symbol_info("EURCAD")
    print(symbol_info)
    print("EURCAD: currency_base =",symbol_info.currency_base,"  currency_profit =",sy
    print()

    # get symbol properties in the form of a dictionary
    print("Show symbol_info()._asdict():")
    symbol_info_dict = symbol_info._asdict()
    for prop in symbol_info_dict:
        print("  {}={}".format(prop, symbol_info_dict[prop]))
    print()
 
    # convert the dictionary into DataFrame and print
    df=pd.DataFrame(list(symbol_info_dict.items()),columns=['property','value'])
    print("symbol_info_dict() as dataframe:")
    print(df)
 
# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
 
 
Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, ses
EURCAD: currency_base = EUR   currency_profit = CAD   currency_margin = EUR
 
Show symbol_info()._asdict():
  custom=False
  chart_mode=0
  select=True
  visible=True
  session_deals=0
  session_buy_orders=0
  session_sell_orders=0
  volume=0
  volumehigh=0
  volumelow=0
  time=1585217595
  digits=5
  spread=39
  spread_float=True
  ticks_bookdepth=10
  trade_calc_mode=0
  trade_mode=4
  start_time=0
  expiration_time=0
  trade_stops_level=0
  trade_freeze_level=0
  trade_exemode=1
  swap_mode=1
  swap_rollover3days=3
  margin_hedged_use_leg=False
  expiration_mode=7
  filling_mode=1
  order_mode=127
  order_gtc_mode=0
```

```
option_mode=0
option_right=0
bid=1.55192
bidhigh=1.55842
bidlow=1.5419800000000001
ask=1.5523099999999999
askhigh=1.55915
asklow=1.5436299999999998
last=0.0
lasthigh=0.0
lastlow=0.0
volume_real=0.0
volumehigh_real=0.0
volumelow_real=0.0
option_strike=0.0
point=1e-05
trade_tick_value=0.7043642408362214
trade_tick_value_profit=0.7043642408362214
trade_tick_value_loss=0.7044535553770941
trade_tick_size=1e-05
trade_contract_size=100000.0
trade_accrued_interest=0.0
trade_face_value=0.0
trade_liquidity_rate=0.0
volume_min=0.01
volume_max=500.0
volume_step=0.01
volume_limit=0.0
swap_long=-1.1
swap_short=-0.9
margin_initial=0.0
margin_maintenance=0.0
session_volume=0.0
session_turnover=0.0
session_interest=0.0
session_buy_orders_volume=0.0
session_sell_orders_volume=0.0
session_open=0.0
session_close=0.0
session_aw=0.0
session_price_settlement=0.0
session_price_limit_min=0.0
session_price_limit_max=0.0
margin_hedged=100000.0
price_change=0.0
price_volatility=0.0
price_theoretical=0.0
price_greeks_delta=0.0
price_greeks_theta=0.0
```

```
  price_greeks_gamma=0.0
  price_greeks_vega=0.0
  price_greeks_rho=0.0
  price_greeks_omega=0.0
  price_sensitivity=0.0
  basis=
  category=
  currency_base=EUR
  currency_profit=CAD
  currency_margin=EUR
  bank=
  description=Euro vs Canadian Dollar
  exchange=
  formula=
  isin=
  name=EURCAD
  page=http://www.google.com/finance?q=EURCAD
  path=Forex\EURCAD

symbol_info_dict() as dataframe:
         property                                  value
0          custom                                  False
1      chart_mode                                      0
2          select                                   True
3         visible                                   True
4   session_deals                                      0
..            ...                                    ...
91        formula
92           isin
93           name                                 EURCAD
94           page  http://www.google.com/finance?q=EURCAD
95           path                           Forex\EURCAD

[96 rows x 2 columns]
```

See also

symbol_info

market_book_add

Subscribes the MetaTrader 5 terminal to the Market Depth change events for a specified symbol.

```
market_book_add(
   symbol      // financial instrument name
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

Return Value

True if successful, otherwise - False.

Note

The function is similar to MarketBookAdd.

See also

market_book_get, market_book_release, Market Depth structure

market_book_get

Returns a tuple from BookInfo featuring Market Depth entries for the specified symbol.

```
market_book_get(
   symbol      // financial instrument name
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

Return Value

Returns the Market Depth content as a tuple from BookInfo entries featuring order type, price and volume in lots. BookInfo is similar to the MqlBookInfo structure.

Return None in case of an error. The info on the error can be obtained using last_error().

Note

The subscription to the Market Depth change events should be preliminarily performed using the market_book_add() function.

The function is similar to MarketBookGet.

Example:

```
import MetaTrader5 as mt5
import time
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print("")

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
   # shut down connection to the MetaTrader 5 terminal
    mt5.shutdown()
    quit()

# subscribe to market depth updates for EURUSD (Depth of Market)
if mt5.market_book_add('EURUSD'):
  # get the market depth data 10 times in a loop
   for i in range(10):
        # get the market depth content (Depth of Market)
        items = mt5.market_book_get('EURUSD')
        # display the entire market depth 'as is' in a single string
        print(items)
        # now display each order separately for more clarity
        if items:
            for it in items:
                # order content
                print(it._asdict())
        # pause for 5 seconds before the next request of the market depth data
        time.sleep(5)
  # cancel the subscription to the market depth updates (Depth of Market)
   mt5.market_book_release('EURUSD')
else:
    print("mt5.market_book_add('EURUSD') failed, error code =",mt5.last_error())

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()



Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.34

(BookInfo(type=1, price=1.20038, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20038, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20032, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.2003, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20028, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20026, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20025, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20023, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20017, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.2004299999999999, volume=250, volume_dbl=250.0), BookInfo(ty
{'type': 1, 'price': 1.2004299999999999, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20037, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20036, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20034, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20031, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20029, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20028, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20022, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.2004299999999999, volume=250, volume_dbl=250.0), BookInfo(ty
{'type': 1, 'price': 1.2004299999999999, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20037, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20036, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20034, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20031, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20029, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20028, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20022, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20036, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20036, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20029, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20028, 'volume': 50, 'volume_dbl': 50.0}
```

```
{'type': 1, 'price': 1.20026, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20023, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20022, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20021, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20014, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20035, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20035, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20029, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.20027, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20025, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20023, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20022, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20021, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20014, 'volume': 250, 'volume_dbl': 250.0}
(BookInfo(type=1, price=1.20037, volume=250, volume_dbl=250.0), BookInfo(type=1, price
{'type': 1, 'price': 1.20037, 'volume': 250, 'volume_dbl': 250.0}
{'type': 1, 'price': 1.20031, 'volume': 100, 'volume_dbl': 100.0}
{'type': 1, 'price': 1.2003, 'volume': 50, 'volume_dbl': 50.0}
{'type': 1, 'price': 1.20028, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20025, 'volume': 36, 'volume_dbl': 36.0}
{'type': 2, 'price': 1.20023, 'volume': 50, 'volume_dbl': 50.0}
{'type': 2, 'price': 1.20022, 'volume': 100, 'volume_dbl': 100.0}
{'type': 2, 'price': 1.20016, 'volume': 250, 'volume_dbl': 250.0}
```

See also

market_book_add, market_book_release, Market Depth structure

market_book_release

Cancels subscription of the MetaTrader 5 terminal to the Market Depth change events for a specified symbol.

```
market_book_release(
   symbol      // financial instrument name
)
```

symbol

[in] Financial instrument name. Required unnamed parameter.

Return Value

True if successful, otherwise - False.

Note

The function is similar to MarketBookRelease.

See also

market_book_add, market_book_get, Market Depth structure

copy_rates_from

Get bars from the MetaTrader 5 terminal starting from the specified date.

```
copy_rates_from(
   symbol,       // symbol name
   timeframe,    // timeframe
   date_from,    // initial bar open date
   count         // number of bars
   )
```

Parameters

symbol

[in] Financial instrument name, for example, "EURUSD". Required unnamed parameter.

timeframe

[in] Timeframe the bars are requested for. Set by a value from the TIMEFRAME enumeration. Required unnamed parameter.

date_from

[in] Date of opening of the first bar from the requested sample. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

count

[in] Number of bars to receive. Required unnamed parameter.

Return Value

Returns bars as the numpy array with the named time, open, high, low, close, tick_volume, spread and real_volume columns. Return None in case of an error. The info on the error can be obtained using last_error().

Note

See the CopyRates() function for more information.

Only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar is always less or equal to the specified one.

MetaTrader 5 terminal provides bars only within a history available to a user on charts. The number of bars available to users is set in the "Max. bars in chart" parameter.

When creating the 'datetime' object, Python uses the local time zone, while MetaTrader 5 stores tick and bar open time in UTC time zone (without the shift). Therefore, 'datetime' should be created in UTC time for executing functions that use time. Data received from the MetaTrader 5 terminal has UTC time.

TIMEFRAME is an enumeration with possible chart period values

ID	Description
TIMEFRAME_M1	1 minute

ID	Description
TIMEFRAME_M2	2 minutes
TIMEFRAME_M3	3 minutes
TIMEFRAME_M4	4 minutes
TIMEFRAME_M5	5 minutes
TIMEFRAME_M6	6 minutes
TIMEFRAME_M10	10 minutes
TIMEFRAME_M12	12 minutes
TIMEFRAME_M12	15 minutes
TIMEFRAME_M20	20 minutes
TIMEFRAME_M30	30 minutes
TIMEFRAME_H1	1 hour
TIMEFRAME_H2	2 hours
TIMEFRAME_H3	3 hours
TIMEFRAME_H4	4 hours
TIMEFRAME_H6	6 hours
TIMEFRAME_H8	8 hours
TIMEFRAME_H12	12 hours
TIMEFRAME_D1	1 day
TIMEFRAME_W1	1 week
TIMEFRAME_MN1	1 month

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz
```

```
# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' object in UTC time zone to avoid the implementation of a local ti
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
# get 10 EURUSD H4 bars starting from 01.10.2020 in UTC time zone
rates = mt5.copy_rates_from("EURUSD", mt5.TIMEFRAME_H4, utc_from, 10)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
# display each element of obtained data in a new line
print("Display obtained data 'as is'")
for rate in rates:
    print(rate)

# create DataFrame out of the obtained data
rates_frame = pd.DataFrame(rates)
# convert time in seconds into the datetime format
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with data")
print(rates_frame)



Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Display obtained data 'as is'
(1578484800, 1.11382, 1.11385, 1.1111, 1.11199, 9354, 1, 0)
(1578499200, 1.11199, 1.11308, 1.11086, 1.11179, 10641, 1, 0)
(1578513600, 1.11178, 1.11178, 1.11016, 1.11053, 4806, 1, 0)
(1578528000, 1.11053, 1.11193, 1.11033, 1.11173, 3480, 1, 0)
(1578542400, 1.11173, 1.11189, 1.11126, 1.11182, 2236, 1, 0)
(1578556800, 1.11181, 1.11203, 1.10983, 1.10993, 7984, 1, 0)
(1578571200, 1.10994, 1.11173, 1.10965, 1.11148, 7406, 1, 0)
(1578585600, 1.11149, 1.11149, 1.10923, 1.11046, 7468, 1, 0)
(1578600000, 1.11046, 1.11097, 1.11033, 1.11051, 3450, 1, 0)
(1578614400, 1.11051, 1.11093, 1.11017, 1.11041, 2448, 1, 0)

Display dataframe with data
                 time     open     high      low    close  tick_volume  spread  real_
0 2020-01-08 12:00:00  1.11382  1.11385  1.11110  1.11199         9354       1
```

```
1 2020-01-08 16:00:00  1.11199  1.11308  1.11086  1.11179        10641        1
2 2020-01-08 20:00:00  1.11178  1.11178  1.11016  1.11053         4806        1
3 2020-01-09 00:00:00  1.11053  1.11193  1.11033  1.11173         3480        1
4 2020-01-09 04:00:00  1.11173  1.11189  1.11126  1.11182         2236        1
5 2020-01-09 08:00:00  1.11181  1.11203  1.10983  1.10993         7984        1
6 2020-01-09 12:00:00  1.10994  1.11173  1.10965  1.11148         7406        1
7 2020-01-09 16:00:00  1.11149  1.11149  1.10923  1.11046         7468        1
8 2020-01-09 20:00:00  1.11046  1.11097  1.11033  1.11051         3450        1
9 2020-01-10 00:00:00  1.11051  1.11093  1.11017  1.11041         2448        1
```

See also

CopyRates, copy_rates_from_pos, copy_rates_range, copy_ticks_from, copy_ticks_range

copy_rates_from_pos

Get bars from the MetaTrader 5 terminal starting from the specified index.

```
copy_rates_from_pos(
   symbol,       // symbol name
   timeframe,    // timeframe
   start_pos,    // initial bar index
   count         // number of bars
   )
```

Parameters

symbol

[in] Financial instrument name, for example, "EURUSD". Required unnamed parameter.

timeframe

[in] Timeframe the bars are requested for. Set by a value from the TIMEFRAME enumeration. Required unnamed parameter.

start_pos

[in] Initial index of the bar the data are requested from. The numbering of bars goes from present to past. Thus, the zero bar means the current one. Required unnamed parameter.

count

[in] Number of bars to receive. Required unnamed parameter.

Return Value

Returns bars as the numpy array with the named time, open, high, low, close, tick_volume, spread and real_volume columns. Returns None in case of an error. The info on the error can be obtained using last_error().

Note

See the CopyRates() function for more information.

MetaTrader 5 terminal provides bars only within a history available to a user on charts. The number of bars available to users is set in the "Max. bars in chart" parameter.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
```

```
# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get 10 GBPUSD D1 bars from the current day
rates = mt5.copy_rates_from_pos("GBPUSD", mt5.TIMEFRAME_D1, 0, 10)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
# display each element of obtained data in a new line
print("Display obtained data 'as is'")
for rate in rates:
    print(rate)

# create DataFrame out of the obtained data
rates_frame = pd.DataFrame(rates)
# convert time in seconds into the datetime format
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with data")
print(rates_frame)

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Display obtained data 'as is'
(1581552000, 1.29568, 1.30692, 1.29441, 1.30412, 68228, 0, 0)
(1581638400, 1.30385, 1.30631, 1.3001, 1.30471, 56498, 0, 0)
(1581897600, 1.30324, 1.30536, 1.29975, 1.30039, 49400, 0, 0)
(1581984000, 1.30039, 1.30486, 1.29705, 1.29952, 62288, 0, 0)
(1582070400, 1.29952, 1.3023, 1.29075, 1.29187, 57909, 0, 0)
(1582156800, 1.29186, 1.29281, 1.28489, 1.28792, 61033, 0, 0)
(1582243200, 1.28802, 1.29805, 1.28746, 1.29566, 66386, 0, 0)
(1582502400, 1.29426, 1.29547, 1.28865, 1.29283, 66933, 0, 0)
(1582588800, 1.2929, 1.30178, 1.29142, 1.30037, 80121, 0, 0)
(1582675200, 1.30036, 1.30078, 1.29136, 1.29374, 49286, 0, 0)

Display dataframe with data
        time     open     high      low    close  tick_volume  spread  real_volume
0 2020-02-13  1.29568  1.30692  1.29441  1.30412        68228       0            0
1 2020-02-14  1.30385  1.30631  1.30010  1.30471        56498       0            0
2 2020-02-17  1.30324  1.30536  1.29975  1.30039        49400       0            0
3 2020-02-18  1.30039  1.30486  1.29705  1.29952        62288       0            0
4 2020-02-19  1.29952  1.30230  1.29075  1.29187        57909       0            0
5 2020-02-20  1.29186  1.29281  1.28489  1.28792        61033       0            0
6 2020-02-21  1.28802  1.29805  1.28746  1.29566        66386       0            0
```

```
7 2020-02-24  1.29426  1.29547  1.28865  1.29283        66933        0             0
8 2020-02-25  1.29290  1.30178  1.29142  1.30037        80121        0             0
9 2020-02-26  1.30036  1.30078  1.29136  1.29374        49286        0             0
```

See also

CopyRates, copy_rates_from, copy_rates_range, copy_ticks_from, copy_ticks_range

copy_rates_range

Get bars in the specified date range from the MetaTrader 5 terminal.

```
copy_rates_range(
   symbol,       // symbol name
   timeframe,    // timeframe
   date_from,    // date the bars are requested from
   date_to       // date, up to which the bars are requested
   )
```

Parameters

symbol

[in] Financial instrument name, for example, "EURUSD". Required unnamed parameter.

timeframe

[in] Timeframe the bars are requested for. Set by a value from the TIMEFRAME enumeration. Required unnamed parameter.

date_from

[in] Date the bars are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Bars with the open time >= date_from are returned. Required unnamed parameter.

date_to

[in] Date, up to which the bars are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Bars with the open time <= date_to are returned. Required unnamed parameter.

Return Value

Returns bars as the numpy array with the named time, open, high, low, close, tick_volume, spread and real_volume columns. Returns None in case of an error. The info on the error can be obtained using last_error().

Note

See the CopyRates() function for more information.

MetaTrader 5 terminal provides bars only within a history available to a user on charts. The number of bars available to users is set in the "Max. bars in chart" parameter.

When creating the 'datetime' object, Python uses the local time zone, while MetaTrader 5 stores tick and bar open time in UTC time zone (without the shift). Therefore, 'datetime' should be created in UTC time for executing functions that use time. Data received from the MetaTrader 5 terminal has UTC time.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' objects in UTC time zone to avoid the implementation of a local ti
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
utc_to = datetime(2020, 1, 11, hour = 13, tzinfo=timezone)
# get bars from USDJPY M5 within the interval of 2020.01.10 00:00 - 2020.01.11 13:00
rates = mt5.copy_rates_range("USDJPY", mt5.TIMEFRAME_M5, utc_from, utc_to)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display each element of obtained data in a new line
print("Display obtained data 'as is'")
counter=0
for rate in rates:
    counter+=1
    if counter<=10:
        print(rate)

# create DataFrame out of the obtained data
rates_frame = pd.DataFrame(rates)
# convert time in seconds into the 'datetime' format
rates_frame['time']=pd.to_datetime(rates_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with data")
print(rates_frame.head(10))


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Display obtained data 'as is'
(1578614400, 109.513, 109.527, 109.505, 109.521, 43, 2, 0)
(1578614700, 109.521, 109.549, 109.518, 109.543, 215, 8, 0)
(1578615000, 109.543, 109.543, 109.466, 109.505, 98, 10, 0)
(1578615300, 109.504, 109.534, 109.502, 109.517, 155, 8, 0)
(1578615600, 109.517, 109.539, 109.513, 109.527, 71, 4, 0)
(1578615900, 109.526, 109.537, 109.484, 109.52, 106, 9, 0)
(1578616200, 109.52, 109.524, 109.508, 109.51, 205, 7, 0)
(1578616500, 109.51, 109.51, 109.491, 109.496, 44, 8, 0)
```

```
(1578616800, 109.496, 109.509, 109.487, 109.5, 85, 5, 0)
(1578617100, 109.5, 109.504, 109.487, 109.489, 82, 7, 0)

Display dataframe with data
                 time     open     high      low    close  tick_volume  spread  real_
0 2020-01-10 00:00:00  109.513  109.527  109.505  109.521           43       2
1 2020-01-10 00:05:00  109.521  109.549  109.518  109.543          215       8
2 2020-01-10 00:10:00  109.543  109.543  109.466  109.505           98      10
3 2020-01-10 00:15:00  109.504  109.534  109.502  109.517          155       8
4 2020-01-10 00:20:00  109.517  109.539  109.513  109.527           71       4
5 2020-01-10 00:25:00  109.526  109.537  109.484  109.520          106       9
6 2020-01-10 00:30:00  109.520  109.524  109.508  109.510          205       7
7 2020-01-10 00:35:00  109.510  109.510  109.491  109.496           44       8
8 2020-01-10 00:40:00  109.496  109.509  109.487  109.500           85       5
9 2020-01-10 00:45:00  109.500  109.504  109.487  109.489           82       7
```

See also

CopyRates, copy_rates_from, copy_rates_range, copy_ticks_from, copy_ticks_range

copy_ticks_from

Get ticks from the MetaTrader 5 terminal starting from the specified date.

```
copy_ticks_from(
   symbol,       // symbol name
   date_from,    // date the ticks are requested from
   count,        // number of requested ticks
   flags         // combination of flags defining the type of requested ticks
   )
```

Parameters

symbol

[in] Financial instrument name, for example, "EURUSD". Required unnamed parameter.

from

[in] Date the ticks are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

count

[in] Number of ticks to receive. Required unnamed parameter.

flags

[in] A flag to define the type of the requested ticks. COPY_TICKS_INFO – ticks with Bid and/or Ask changes, COPY_TICKS_TRADE – ticks with changes in Last and Volume, COPY_TICKS_ALL – all ticks. Flag values are described in the COPY_TICKS enumeration. Required unnamed parameter.

Return Value

Returns ticks as the numpy array with the named time, bid, ask, last and flags columns. The 'flags' value can be a combination of flags from the TICK_FLAG enumeration. Return None in case of an error. The info on the error can be obtained using last_error().

Note

See the CopyTicks function for more information.

When creating the 'datetime' object, Python uses the local time zone, while MetaTrader 5 stores tick and bar open time in UTC time zone (without the shift). Therefore, 'datetime' should be created in UTC time for executing functions that use time. Data received from the MetaTrader 5 terminal has UTC time.

COPY_TICKS defines the types of ticks that can be requested using the copy_ticks_from() and copy_ticks_range() functions.

ID	Description
COPY_TICKS_ALL	all ticks
COPY_TICKS_INFO	ticks containing Bid and/or Ask price changes
COPY_TICKS_TRADE	ticks containing Last and/or Volume price changes

TICK_FLAG defines possible flags for ticks. These flags are used to describe ticks obtained by the copy_ticks_from() and copy_ticks_range() functions.

ID	Description
TICK_FLAG_BID	Bid price changed
TICK_FLAG_ASK	Ask price changed
TICK_FLAG_LAST	Last price changed
TICK_FLAG_VOLUME	Volume changed
TICK_FLAG_BUY	last Buy price changed
TICK_FLAG_SELL	last Sell price changed

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' object in UTC time zone to avoid the implementation of a local ti
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
# request 100 000 EURUSD ticks starting from 10.01.2019 in UTC time zone
ticks = mt5.copy_ticks_from("EURUSD", utc_from, 100000, mt5.COPY_TICKS_ALL)
print("Ticks received:",len(ticks))

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display data on each tick on a new line
print("Display obtained ticks 'as is'")
count = 0
for tick in ticks:
    count+=1
    print(tick)
    if count >= 10:
        break

# create DataFrame out of the obtained data
ticks_frame = pd.DataFrame(ticks)
# convert time in seconds into the datetime format
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with ticks")
print(ticks_frame.head(10))

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Ticks received: 100000
Display obtained ticks 'as is'
(1578614400, 1.11051, 1.11069, 0., 0, 1578614400987, 134, 0.)
(1578614402, 1.11049, 1.11067, 0., 0, 1578614402025, 134, 0.)
(1578614404, 1.1105, 1.11066, 0., 0, 1578614404057, 134, 0.)
(1578614404, 1.11049, 1.11067, 0., 0, 1578614404344, 134, 0.)
(1578614412, 1.11052, 1.11064, 0., 0, 1578614412106, 134, 0.)
(1578614418, 1.11039, 1.11051, 0., 0, 1578614418265, 134, 0.)
(1578614418, 1.1104, 1.1105, 0., 0, 1578614418905, 134, 0.)
(1578614419, 1.11039, 1.11051, 0., 0, 1578614419519, 134, 0.)
(1578614456, 1.11037, 1.11065, 0., 0, 1578614456011, 134, 0.)
(1578614456, 1.11039, 1.11051, 0., 0, 1578614456015, 134, 0.)

Display dataframe with ticks
                 time      bid      ask  last  volume       time_msc  flags  volume_re
0 2020-01-10 00:00:00  1.11051  1.11069   0.0       0  1578614400987    134
1 2020-01-10 00:00:02  1.11049  1.11067   0.0       0  1578614402025    134
2 2020-01-10 00:00:04  1.11050  1.11066   0.0       0  1578614404057    134
3 2020-01-10 00:00:04  1.11049  1.11067   0.0       0  1578614404344    134
4 2020-01-10 00:00:12  1.11052  1.11064   0.0       0  1578614412106    134
5 2020-01-10 00:00:18  1.11039  1.11051   0.0       0  1578614418265    134
6 2020-01-10 00:00:18  1.11040  1.11050   0.0       0  1578614418905    134
7 2020-01-10 00:00:19  1.11039  1.11051   0.0       0  1578614419519    134
8 2020-01-10 00:00:56  1.11037  1.11065   0.0       0  1578614456011    134
9 2020-01-10 00:00:56  1.11039  1.11051   0.0       0  1578614456015    134
```

See also

CopyRates, copy_rates_from_pos, copy_rates_range, copy_ticks_from, copy_ticks_range

copy_ticks_range

Get ticks for the specified date range from the MetaTrader 5 terminal.

```
copy_ticks_range(
   symbol,        // symbol name
   date_from,     // date the ticks are requested from
   date_to,       // date, up to which the ticks are requested
   flags          // combination of flags defining the type of requested ticks
   )
```

Parameters

symbol

[in] Financial instrument name, for example, "EURUSD". Required unnamed parameter.

date_from

[in] Date the ticks are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

date_to

[in] Date, up to which the ticks are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

flags

[in] A flag to define the type of the requested ticks. COPY_TICKS_INFO – ticks with Bid and/or Ask changes, COPY_TICKS_TRADE – ticks with changes in Last and Volume, COPY_TICKS_ALL – all ticks. Flag values are described in the COPY_TICKS enumeration. Required unnamed parameter.

Return Value

Returns ticks as the numpy array with the named time, bid, ask, last and flags columns. The 'flags' value can be a combination of flags from the TICK_FLAG enumeration. Return None in case of an error. The info on the error can be obtained using last_error().

Note

See the CopyTicks function for more information.

When creating the 'datetime' object, Python uses the local time zone, while MetaTrader 5 stores tick and bar open time in UTC time zone (without the shift). Therefore, 'datetime' should be created in UTC time for executing functions that use time. The data obtained from MetaTrader 5 have UTC time, but Python applies the local time shift again when trying to print them. Thus, the obtained data should also be corrected for visual presentation.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
```

```
# import the 'pandas' module for displaying data obtained in the tabular form
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# import pytz module for working with time zone
import pytz

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# set time zone to UTC
timezone = pytz.timezone("Etc/UTC")
# create 'datetime' objects in UTC time zone to avoid the implementation of a local t
utc_from = datetime(2020, 1, 10, tzinfo=timezone)
utc_to = datetime(2020, 1, 11, tzinfo=timezone)
# request AUDUSD ticks within 11.01.2020 - 11.01.2020
ticks = mt5.copy_ticks_range("AUDUSD", utc_from, utc_to, mt5.COPY_TICKS_ALL)
print("Ticks received:",len(ticks))

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

# display data on each tick on a new line
print("Display obtained ticks 'as is'")
count = 0
for tick in ticks:
    count+=1
    print(tick)
    if count >= 10:
        break

# create DataFrame out of the obtained data
ticks_frame = pd.DataFrame(ticks)
# convert time in seconds into the datetime format
ticks_frame['time']=pd.to_datetime(ticks_frame['time'], unit='s')

# display data
print("\nDisplay dataframe with ticks")
print(ticks_frame.head(10))


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Ticks received: 37008
Display obtained ticks 'as is'
(1578614400, 0.68577, 0.68594, 0., 0, 1578614400820, 134, 0.)
(1578614401, 0.68578, 0.68594, 0., 0, 1578614401128, 130, 0.)
(1578614401, 0.68575, 0.68594, 0., 0, 1578614401128, 130, 0.)
(1578614411, 0.68576, 0.68594, 0., 0, 1578614411388, 130, 0.)
(1578614411, 0.68575, 0.68594, 0., 0, 1578614411560, 130, 0.)
(1578614414, 0.68576, 0.68595, 0., 0, 1578614414973, 134, 0.)
(1578614430, 0.68576, 0.68594, 0., 0, 1578614430188, 4, 0.)
(1578614450, 0.68576, 0.68595, 0., 0, 1578614450408, 4, 0.)
```

```
(1578614450, 0.68576, 0.68594, 0., 0, 1578614450519, 4, 0.)
(1578614456, 0.68575, 0.68594, 0., 0, 1578614456363, 130, 0.)

Display dataframe with ticks
                 time      bid      ask  last  volume       time_msc  flags  volume_re
0 2020-01-10 00:00:00  0.68577  0.68594   0.0       0  1578614400820    134          (
1 2020-01-10 00:00:01  0.68578  0.68594   0.0       0  1578614401128    130          (
2 2020-01-10 00:00:01  0.68575  0.68594   0.0       0  1578614401128    130          (
3 2020-01-10 00:00:11  0.68576  0.68594   0.0       0  1578614411388    130          (
4 2020-01-10 00:00:11  0.68575  0.68594   0.0       0  1578614411560    130          (
5 2020-01-10 00:00:14  0.68576  0.68595   0.0       0  1578614414973    134          (
6 2020-01-10 00:00:30  0.68576  0.68594   0.0       0  1578614430188      4          (
7 2020-01-10 00:00:50  0.68576  0.68595   0.0       0  1578614450408      4          (
8 2020-01-10 00:00:50  0.68576  0.68594   0.0       0  1578614450519      4          (
9 2020-01-10 00:00:56  0.68575  0.68594   0.0       0  1578614456363    130          (
```

See also

CopyRates, copy_rates_from_pos, copy_rates_range, copy_ticks_from, copy_ticks_range

orders_total

Get the number of active orders.

```
orders_total()
```

Return Value

Integer value.

Note

The function is similar to OrdersTotal.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# check the presence of active orders
orders=mt5.orders_total()
if orders>0:
    print("Total orders=",orders)
else:
    print("Orders not found")

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

orders_get, positions_total

orders_get

Get active orders with the ability to filter by symbol or ticket. There are three call options.

Call without parameters. Return active orders on all symbols.

```
orders_get()
```

Call specifying a symbol active orders should be received for.

```
orders_get(
   symbol="SYMBOL"       // symbol name
)
```

Call specifying a group of symbols active orders should be received for.

```
orders_get(
   group="GROUP"         // filter for selecting orders for symbols
)
```

Call specifying the order ticket.

```
orders_get(
   ticket=TICKET         // ticket
)
```

symbol="SYMBOL"

[in] Symbol name. Optional named parameter. If a symbol is specified, the *ticket* parameter is ignored.

group="GROUP"

[in] The filter for arranging a group of necessary symbols. Optional named parameter. If the group is specified, the function returns only active orders meeting a specified criteria for a symbol name.

ticket=TICKET

[in] Order ticket (ORDER_TICKET). Optional named parameter.

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function allows receiving all active orders within one call similar to the OrdersTotal and OrderSelect tandem.

The *group* parameter allows sorting out orders by symbols. '*' can be used at the beginning and the end of a string.

The *group* parameter may contain several comma separated conditions. A condition can be set as a mask using '*'. The logical negation symbol '!' can be used for an exclusion. All conditions are applied sequentially, which means conditions of including to a group should be specified first followed by an exclusion condition. For example, *group*="*, !EUR" means that orders for all symbols should be selected first and the ones containing "EUR" in symbol names should be excluded afterwards.

Example:

```
import MetaTrader5 as mt5
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# display data on active orders on GBPUSD
orders=mt5.orders_get(symbol="GBPUSD")
if orders is None:
    print("No orders on GBPUSD, error code={}".format(mt5.last_error()))
else:
    print("Total orders on GBPUSD:",len(orders))
    # display all active orders
    for order in orders:
        print(order)
print()

# get the list of orders on symbols whose names contain "*GBP*"
gbp_orders=mt5.orders_get(group="*GBP*")
if gbp_orders is None:
    print("No orders with group=\"*GBP*\", error code={}".format(mt5.last_error()))
else:
    print("orders_get(group=\"*GBP*\")={}".format(len(gbp_orders)))
    # display these orders as a table using pandas.DataFrame
    df=pd.DataFrame(list(gbp_orders),columns=gbp_orders[0]._asdict().keys())
    df.drop(['time_done', 'time_done_msc', 'position_id', 'position_by_id', 'reason',
    df['time_setup'] = pd.to_datetime(df['time_setup'], unit='s')
    print(df)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Total orders on GBPUSD: 2
TradeOrder(ticket=554733548, time_setup=1585153667, time_setup_msc=1585153667718, time
TradeOrder(ticket=554733621, time_setup=1585153671, time_setup_msc=1585153671419, time
orders_get(group="*GBP*")=4
      ticket          time_setup  time_setup_msc  time_expiration  type  type_time  ty
0  554733548 2020-03-25 16:27:47   1585153667718                0     3          0
1  554733621 2020-03-25 16:27:51   1585153671419                0     2          0
2  554746664 2020-03-25 16:38:14   1585154294401                0     3          0
3  554746710 2020-03-25 16:38:17   1585154297022                0     2          0
```

See also

orders_total, positions_get

order_calc_margin

Return margin in the account currency to perform a specified trading operation.

```
order_calc_margin(
   action,       // order type (ORDER_TYPE_BUY or ORDER_TYPE_SELL)
   symbol,       // symbol name
   volume,       // volume
   price         // open price
   )
```

Parameters

action

[in] Order type taking values from the ORDER_TYPE enumeration. Required unnamed parameter.

symbol

[in] Financial instrument name. Required unnamed parameter.

volume

[in] Trading operation volume. Required unnamed parameter.

price

[in] Open price. Required unnamed parameter.

Return Value

Real value if successful, otherwise None. The error info can be obtained using last_error().

Note

The function allows estimating the margin necessary for a specified order type on the current account and in the current market environment without considering the current pending orders and open positions. The function is similar to OrderCalcMargin.

ORDER_TYPE

ID	Description
ORDER_TYPE_BUY	Market buy order
ORDER_TYPE_SELL	Market sell order
ORDER_TYPE_BUY_LIMIT	Buy Limit pending order
ORDER_TYPE_SELL_LIMIT	Sell Limit pending order
ORDER_TYPE_BUY_STOP	Buy Stop pending order
ORDER_TYPE_SELL_STOP	Sell Stop pending order
ORDER_TYPE_BUY_STOP_LIMIT	Upon reaching the order price, Buy Limit pending order is placed at StopLimit price
ORDER_TYPE_SELL_STOP_LIMIT	Upon reaching the order price, Sell Limit pending order is placed at StopLimit price

ID	Description
ORDER_TYPE_CLOSE_BY	Order for closing a position by an opposite one

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# arrange the symbol list
symbols=("EURUSD","GBPUSD","USDJPY", "USDCHF","EURJPY","GBPJPY")
print("Symbols to check margin:", symbols)
action=mt5.ORDER_TYPE_BUY
lot=0.1
for symbol in symbols:
    symbol_info=mt5.symbol_info(symbol)
    if symbol_info is None:
        print(symbol,"not found, skipped")
        continue
    if not symbol_info.visible:
        print(symbol, "is not visible, trying to switch on")
        if not mt5.symbol_select(symbol,True):
            print("symbol_select({}}) failed, skipped",symbol)
            continue
    ask=mt5.symbol_info_tick(symbol).ask
    margin=mt5.order_calc_margin(action,symbol,lot,ask)
    if margin != None:
        print("   {} buy {} lot margin: {} {}".format(symbol,lot,margin,account_curren
    else:
        print("order_calc_margin failed: , error code =", mt5.last_error())

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currency: USD

Symbols to check margin: ('EURUSD', 'GBPUSD', 'USDJPY', 'USDCHF', 'EURJPY', 'GBPJPY')
   EURUSD buy 0.1 lot margin: 109.91 USD
   GBPUSD buy 0.1 lot margin: 122.73 USD
```

```
    USDJPY buy 0.1 lot margin: 100.0 USD
    USDCHF buy 0.1 lot margin: 100.0 USD
    EURJPY buy 0.1 lot margin: 109.91 USD
    GBPJPY buy 0.1 lot margin: 122.73 USD
```

See also

order_calc_profit, order_check

order_calc_profit

Return profit in the account currency for a specified trading operation.

```
order_calc_profit(
   action,            // order type (ORDER_TYPE_BUY or ORDER_TYPE_SELL)
   symbol,            // symbol name
   volume,            // volume
   price_open,        // open price
   price_close        // close price
   );
```

Parameters

action

[in] Order type may take one of the two ORDER_TYPE enumeration values: ORDER_TYPE_BUY or ORDER_TYPE_SELL. Required unnamed parameter.

symbol

[in] Financial instrument name. Required unnamed parameter.

volume

[in] Trading operation volume. Required unnamed parameter.

price_open

[in] Open price. Required unnamed parameter.

price_close

[in] Close price. Required unnamed parameter.

Return Value

Real value if successful, otherwise None. The error info can be obtained using last_error().

Note

The function allows estimating a trading operation result on the current account and in the current trading environment. The function is similar to OrderCalcProfit.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# arrange the symbol list
symbols = ("EURUSD","GBPUSD","USDJPY")
print("Symbols to check margin:", symbols)
# estimate profit for buying and selling
lot=1.0
distance=300
for symbol in symbols:
    symbol_info=mt5.symbol_info(symbol)
    if symbol_info is None:
        print(symbol,"not found, skipped")
        continue
    if not symbol_info.visible:
        print(symbol, "is not visible, trying to switch on")
        if not mt5.symbol_select(symbol,True):
            print("symbol_select({}}) failed, skipped",symbol)
            continue
    point=mt5.symbol_info(symbol).point
    symbol_tick=mt5.symbol_info_tick(symbol)
    ask=symbol_tick.ask
    bid=symbol_tick.bid
    buy_profit=mt5.order_calc_profit(mt5.ORDER_TYPE_BUY,symbol,lot,ask,ask+distance*po
    if buy_profit!=None:
        print("   buy {} {} lot: profit on {} points => {} {}".format(symbol,lot,dista
    else:
        print("order_calc_profit(ORDER_TYPE_BUY) failed, error code =",mt5.last_error
    sell_profit=mt5.order_calc_profit(mt5.ORDER_TYPE_SELL,symbol,lot,bid,bid-distance
    if sell_profit!=None:
        print("   sell {} {} lots: profit on {} points => {} {}".format(symbol,lot,dis
    else:
        print("order_calc_profit(ORDER_TYPE_SELL) failed, error code =",mt5.last_erro
    print()

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currency: USD
Symbols to check margin: ('EURUSD', 'GBPUSD', 'USDJPY')
   buy EURUSD 1.0 lot: profit on 300 points => 300.0 USD
   sell EURUSD 1.0 lot: profit on 300 points => 300.0 USD

   buy GBPUSD 1.0 lot: profit on 300 points => 300.0 USD
   sell GBPUSD 1.0 lot: profit on 300 points => 300.0 USD
```

```
  buy USDJPY 1.0 lot: profit on 300 points => 276.54 USD
  sell USDJPY 1.0 lot: profit on 300 points => 278.09 USD
```

See also

order_calc_margin, order_check

order_check

Check funds sufficiency for performing a required trading operation. Check result are returned as the MqlTradeCheckResult structure.

```
order_check(
   request      // request structure
   );
```

Parameters

request

[in] MqlTradeRequest type structure describing a required trading action. Required unnamed parameter. Example of filling in a request and the enumeration content are described below.

Return Value

Check result as the MqlTradeCheckResult structure. The *request* field in the answer contains the structure of a trading request passed to order_check().

Note

Successful sending of a request **does not entail that the requested trading operation will be executed successfully**. The order_check function is similar to OrderCheck.

TRADE_REQUEST_ACTIONS

ID	Description
TRADE_ACTION_DEAL	Place an order for an instant deal with the specified parameters (set a market order)
TRADE_ACTION_PENDING	Place an order for performing a deal at specified conditions (pending order)
TRADE_ACTION_SLTP	Change open position Stop Loss and Take Profit
TRADE_ACTION_MODIFY	Change parameters of the previously placed trading order
TRADE_ACTION_REMOVE	Remove previously placed pending order
TRADE_ACTION_CLOSE_BY	Close a position by an opposite one

ORDER_TYPE_FILLING

ID	Description
ORDER_FILLING_FOK	This execution policy means that an order can be executed only in the specified volume. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The desired volume can be made up of several available offers.
ORDER_FILLING_IOC	An agreement to execute a deal at the maximum volume available in the market within the volume specified in the order. If the request cannot be filled completely, an order

ID	Description
	with the available volume will be executed, and the remaining volume will be canceled.
ORDER_FILLING_RETURN	This policy is used only for market (ORDER_TYPE_BUY and ORDER_TYPE_SELL), limit and stop limit orders (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT) and only for the symbols with Market or Exchange execution modes. If filled partially, a market or limit order with the remaining volume is not canceled, and is processed further. During activation of the ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT orders, an appropriate limit order ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT with the ORDER_FILLING_RETURN type is created.

ORDER_TYPE_TIME

ID	Description
ORDER_TIME_GTC	The order stays in the queue until it is manually canceled
ORDER_TIME_DAY	The order is active only during the current trading day
ORDER_TIME_SPECIFIED	The order is active until the specified date
ORDER_TIME_SPECIFIED_DAY	The order is active until 23:59:59 of the specified day. If this time appears to be out of a trading session, the expiration is processed at the nearest trading time.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get account currency
account_currency=mt5.account_info().currency
print("Account currency:",account_currency)

# prepare the request structure
symbol="USDJPY"
symbol_info = mt5.symbol_info(symbol)
if symbol_info is None:
    print(symbol, "not found, can not call order_check()")
    mt5.shutdown()
    quit()
```

```
# if the symbol is unavailable in MarketWatch, add it
if not symbol_info.visible:
    print(symbol, "is not visible, trying to switch on")
    if not mt5.symbol_select(symbol,True):
        print("symbol_select({}}) failed, exit",symbol)
        mt5.shutdown()
        quit()

# prepare the request
point=mt5.symbol_info(symbol).point
request = {
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": 1.0,
    "type": mt5.ORDER_TYPE_BUY,
    "price": mt5.symbol_info_tick(symbol).ask,
    "sl": mt5.symbol_info_tick(symbol).ask-100*point,
    "tp": mt5.symbol_info_tick(symbol).ask+100*point,
    "deviation": 10,
    "magic": 234000,
    "comment": "python script",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}

# perform the check and display the result 'as is'
result = mt5.order_check(request)
print(result);
# request the result as a dictionary and display it element by element
result_dict=result._asdict()
for field in result_dict.keys():
    print("   {}={}".format(field,result_dict[field]))
    # if this is a trading request structure, display it element by element as well
    if field=="request":
        traderequest_dict=result_dict[field]._asdict()
        for tradereq_filed in traderequest_dict:
            print("       traderequest: {}={}".format(tradereq_filed,traderequest_dict

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

Account currecy: USD
   retcode=0
   balance=101300.53
   equity=68319.53
   profit=-32981.0
   margin=51193.67
   margin_free=17125.86
   margin_level=133.45308121101692
   comment=Done
   request=TradeRequest(action=1, magic=234000, order=0, symbol='USDJPY', volume=1.0,
       traderequest: action=1
       traderequest: magic=234000
       traderequest: order=0
       traderequest: symbol=USDJPY
       traderequest: volume=1.0
       traderequest: price=108.081
       traderequest: stoplimit=0.0
       traderequest: sl=107.98100000000001
       traderequest: tp=108.181
       traderequest: deviation=10
       traderequest: type=0
       traderequest: type_filling=2
       traderequest: type_time=0
       traderequest: expiration=0
       traderequest: comment=python script
       traderequest: position=0
       traderequest: position_by=0
```

See also

order_send, OrderCheck, Trading operation types, Trading request structure, Structure of the trading request check results, Structure of the trading request result

order_send

Send a request to perform a trading operation from the terminal to the trade server. The function is similar to OrderSend.

```
order_send(
   request      // request structure
   );
```

Parameters

request

[in] MqlTradeRequest type structure describing a required trading action. Required unnamed parameter. Example of filling in a request and the enumeration content are described below.

Return Value

Execution result as the MqlTradeResult structure. The *request* field in the answer contains the structure of a trading request passed to order_send(). The info on the error can be obtained using last_error().

The MqlTradeRequest trading request structure

Field	Description
action	Trading operation type. The value can be one of the values of the TRADE_REQUEST_ACTIONS enumeration
magic	EA ID. Allows arranging the analytical handling of trading orders. Each EA can set a unique ID when sending a trading request
order	Order ticket. Required for modifying pending orders
symbol	The name of the trading instrument, for which the order is placed. Not required when modifying orders and closing positions
volume	Requested volume of a deal in lots. A real volume when making a deal depends on an order execution type.
price	Price at which an order should be executed. The price is not set in case of market orders for instruments of the "Market Execution" (SYMBOL_TRADE_EXECUTION_MARKET) type having the TRADE_ACTION_DEAL type
stoplimit	A price a pending Limit order is set at when the price reaches the 'price' value (this condition is mandatory). The pending order is not passed to the trading system until that moment
sl	A price a Stop Loss order is activated at when the price moves in an unfavorable direction
tp	A price a Take Profit order is activated at when the price moves in a favorable direction
deviation	Maximum acceptable deviation from the requested price, specified in points

Field	Description
type	Order type. The value can be one of the values of the ORDER_TYPE enumeration
type_filling	Order filling type. The value can be one of the ORDER_TYPE_FILLING values
type_time	Order type by expiration. The value can be one of the ORDER_TYPE_TIME values
expiration	Pending order expiration time (for TIME_SPECIFIED type orders)
comment	Comment to an order
position	Position ticket. Fill it when changing and closing a position for its clear identification. Usually, it is the same as the ticket of the order that opened the position.
position_by	Opposite position ticket. It is used when closing a position by an opposite one (opened at the same symbol but in the opposite direction).

Note

A trading request passes several verification stages on the trade server. First, the validity of all the necessary *request* fields is checked. If there are no errors, the server accepts the order for further handling. See the OrderSend function description for the details about executing trading operations.

Example:

```
import time
import MetaTrader5 as mt5

# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ", mt5.__author__)
print("MetaTrader5 package version: ", mt5.__version__)

# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# prepare the buy request structure
symbol = "USDJPY"
symbol_info = mt5.symbol_info(symbol)
if symbol_info is None:
    print(symbol, "not found, can not call order_check()")
    mt5.shutdown()
    quit()

# if the symbol is unavailable in MarketWatch, add it
if not symbol_info.visible:
    print(symbol, "is not visible, trying to switch on")
    if not mt5.symbol_select(symbol,True):
        print("symbol_select({}}) failed, exit",symbol)
        mt5.shutdown()
        quit()

lot = 0.1
point = mt5.symbol_info(symbol).point
price = mt5.symbol_info_tick(symbol).ask
deviation = 20
request = {
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": lot,
    "type": mt5.ORDER_TYPE_BUY,
    "price": price,
    "sl": price - 100 * point,
    "tp": price + 100 * point,
    "deviation": deviation,
    "magic": 234000,
    "comment": "python script open",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}

# send a trading request
result = mt5.order_send(request)
# check the execution result
print("1. order_send(): by {} {} lots at {} with deviation={} points".format(symbol,lo
if result.retcode != mt5.TRADE_RETCODE_DONE:
    print("2. order_send failed, retcode={}".format(result.retcode))
    # request the result as a dictionary and display it element by element
    result_dict=result._asdict()
    for field in result_dict.keys():
        print("   {}={}".format(field,result_dict[field]))
        # if this is a trading request structure, display it element by element as wel
        if field=="request":
            traderequest_dict=result_dict[field]._asdict()
            for tradereq_filed in traderequest_dict:
                print("       traderequest: {}={}".format(tradereq_filed,traderequest_
    print("shutdown() and quit")
    mt5.shutdown()
    quit()

print("2. order_send done, ", result)
print("   opened position with POSITION_TICKET={}".format(result.order))
print("   sleep 2 seconds before closing position #{}".format(result.order))
time.sleep(2)
# create a close request
position_id=result.order
price=mt5.symbol_info_tick(symbol).bid
deviation=20
request={
    "action": mt5.TRADE_ACTION_DEAL,
    "symbol": symbol,
    "volume": lot,
    "type": mt5.ORDER_TYPE_SELL,
    "position": position_id,
    "price": price,
    "deviation": deviation,
    "magic": 234000,
    "comment": "python script close",
    "type_time": mt5.ORDER_TIME_GTC,
    "type_filling": mt5.ORDER_FILLING_RETURN,
}
# send a trading request
result=mt5.order_send(request)
# check the execution result
print("3. close position #{}: sell {} {} lots at {} with deviation={} points".format(p
if result.retcode != mt5.TRADE_RETCODE_DONE:
    print("4. order_send failed, retcode={}".format(result.retcode))
    print("   result",result)
else:
    print("4. position #{} closed, {}".format(position_id,result))
    # request the result as a dictionary and display it element by element
    result_dict=result._asdict()
    for field in result_dict.keys():
        print("   {}={}".format(field,result_dict[field]))
        # if this is a trading request structure, display it element by element as we
        if field=="request":
            traderequest_dict=result_dict[field]._asdict()
            for tradereq_filed in traderequest_dict:
                print("       traderequest: {}={}".format(tradereq_filed,traderequest_

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

Result
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29
1. order_send(): by USDJPY 0.1 lots at 108.023 with deviation=20 points
2. order_send done,  OrderSendResult(retcode=10009, deal=535084512, order=557416535, v
   opened position with POSITION_TICKET=557416535
   sleep 2 seconds before closing position #557416535
3. close position #557416535: sell USDJPY 0.1 lots at 108.018 with deviation=20 points
4. position #557416535 closed, OrderSendResult(retcode=10009, deal=535084631, order=55
   retcode=10009
   deal=535084631
   order=557416654
   volume=0.1
   price=108.015
   bid=108.015
   ask=108.02
   comment=Request executed
   request_id=55
   retcode_external=0
   request=TradeRequest(action=1, magic=234000, order=0, symbol='USDJPY', volume=0.1,
       traderequest: action=1
       traderequest: magic=234000
       traderequest: order=0
       traderequest: symbol=USDJPY
       traderequest: volume=0.1
       traderequest: price=108.018
       traderequest: stoplimit=0.0
       traderequest: sl=0.0
       traderequest: tp=0.0
       traderequest: deviation=20
       traderequest: type=1
       traderequest: type_filling=2
       traderequest: type_time=0
       traderequest: expiration=0
       traderequest: comment=python script close
       traderequest: position=557416535
       traderequest: position_by=0
```

See also

order_check, OrderSend,Trading operation types, Trading request structure, Structure of the trading request check results, Structure of the trading request result

positions_total

Get the number of open positions.

```
positions_total()
```

Return Value

Integer value.

Note

The function is similar to PositionsTotal.

Example:

```
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# check the presence of open positions
positions_total=mt5.positions_total()
if positions_total>0:
    print("Total positions=",positions_total)
else:
    print("Positions not found")

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

positions_get, orders_total

positions_get

Get open positions with the ability to filter by symbol or ticket. There are three call options.

Call without parameters. Return open positions for all symbols.

```
positions_get()
```

Call specifying a symbol open positions should be received for.

```
positions_get(
   symbol="SYMBOL"      // symbol name
)
```

Call specifying a group of symbols open positions should be received for.

```
positions_get(
   group="GROUP"        // filter for selecting positions by symbols
)
```

Call specifying a position ticket.

```
positions_get(
   ticket=TICKET        // ticket
)
```

Parameters

`symbol="SYMBOL"`

[in] Symbol name. Optional named parameter. If a symbol is specified, the *ticket* parameter is ignored.

`group="GROUP"`

[in] The filter for arranging a group of necessary symbols. Optional named parameter. If the group is specified, the function returns only positions meeting a specified criteria for a symbol name.

`ticket=TICKET`

[in] Position ticket (POSITION_TICKET). Optional named parameter.

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function allows receiving all open positions within one call similar to the PositionsTotal and PositionSelect tandem.

The *group* parameter may contain several comma separated conditions. A condition can be set as a mask using '*'. The logical negation symbol '!' can be used for an exclusion. All conditions are applied sequentially, which means conditions of including to a group should be specified first followed by an exclusion condition. For example, *group*="*, !EUR" means that positions for all symbols should be selected first and the ones containing "EUR" in symbol names should be excluded afterwards.

Example:

```
import MetaTrader5 as mt5
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get open positions on USDCHF
positions=mt5.positions_get(symbol="USDCHF")
if positions==None:
    print("No positions on USDCHF, error code={}".format(mt5.last_error()))
elif len(positions)>0:
    print("Total positions on USDCHF =",len(positions))
    # display all open positions
    for position in positions:
        print(position)

# get the list of positions on symbols whose names contain "*USD*"
usd_positions=mt5.positions_get(group="*USD*")
if usd_positions==None:
    print("No positions with group=\"*USD*\", error code={}".format(mt5.last_error()))
elif len(usd_positions)>0:
    print("positions_get(group=\"*USD*\")={}".format(len(usd_positions)))
    # display these positions as a table using pandas.DataFrame
    df=pd.DataFrame(list(usd_positions),columns=usd_positions[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    df.drop(['time_update', 'time_msc', 'time_update_msc', 'external_id'], axis=1, inp
    print(df)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()


Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

positions_get(group="*USD*")=5
      ticket                time  type  magic  identifier  reason  volume  price_open
0  548297723 2020-03-18 15:00:55     1      0   548297723       3    0.01     1.09301
1  548655158 2020-03-18 20:31:26     0      0   548655158       3    0.01     1.08676
```

```
2   548663803 2020-03-18 20:40:04      0        0    548663803         3      0.01       1.08640
3   548847168 2020-03-19 01:10:05      0        0    548847168         3      0.01       1.09545
4   548847194 2020-03-19 01:10:07      0        0    548847194         3      0.02       1.09536
```

See also

positions_total, orders_get

history_orders_total

Get the number of orders in trading history within the specified interval.

```
history_orders_total(
   date_from,    // date the orders are requested from
   date_to       // date, up to which the orders are requested
   )
```

Parameters

date_from

[in] Date the orders are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

date_to

[in] Date, up to which the orders are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

Return Value

Integer value.

Note

The function is similar to HistoryOrdersTotal.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get the number of orders in history
from_date=datetime(2020,1,1)
to_date=datetime.now()
history_orders=mt5.history_orders_total(from_date, datetime.now())
if history_orders>0:
    print("Total history orders=",history_orders)
else:
    print("Orders not found in history")

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

history_orders_get, history_deals_total

history_orders_get

Get orders from trading history with the ability to filter by ticket or position. There are three call options.

Call specifying a time interval. Return all orders falling within the specified interval.

```
history_orders_get(
   date_from,                // date the orders are requested from
   date_to,                  // date, up to which the orders are requested
   group="GROUP"        // filter for selecting orders by symbols
   )
```

Call specifying the order ticket. Return an order with the specified ticket.

```
history_orders_get(
   ticket=TICKET        // order ticket
)
```

Call specifying the position ticket. Return all orders with a position ticket specified in the ORDER_POSITION_ID property.

```
history_orders_get(
   position=POSITION    // position ticket
)
```

Parameters

date_from

[in] Date the orders are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter is specified first.

date_to

[in] Date, up to which the orders are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter is specified second.

group="GROUP"

[in] The filter for arranging a group of necessary symbols. Optional named parameter. If the group is specified, the function returns only orders meeting a specified criteria for a symbol name.

ticket=TICKET

[in] Order ticket that should be received. Optional parameter. If not specified, the filter is not applied.

position=POSITION

[in] Ticket of a position (stored in ORDER_POSITION_ID) all orders should be received for. Optional parameter. If not specified, the filter is not applied.

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function allows receiving all history orders within a specified period in a single call similar to the HistoryOrdersTotal and HistoryOrderSelect tandem.

The *group* parameter may contain several comma separated conditions. A condition can be set as a mask using '*'. The logical negation symbol '!' can be used for an exclusion. All conditions are applied sequentially, which means conditions of including to a group should be specified first followed by an exclusion condition. For example, *group*="*, !EUR" means that deals for all symbols should be selected first and the ones containing "EUR" in symbol names should be excluded afterwards.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get the number of orders in history
from_date=datetime(2020,1,1)
to_date=datetime.now()
history_orders=mt5.history_orders_get(from_date, to_date, group="*GBP*")
if history_orders==None:
    print("No history orders with group=\"*GBP*\", error code={}".format(mt5.last_erro
elif len(history_orders)>0:
    print("history_orders_get({}, {}, group=\"*GBP*\")={}".format(from_date,to_date,le
print()

# display all historical orders by a position ticket
position_id=530218319
position_history_orders=mt5.history_orders_get(position=position_id)
if position_history_orders==None:
    print("No orders with position #{}".format(position_id))
    print("error code =",mt5.last_error())
elif len(position_history_orders)>0:
    print("Total history orders on position #{}: {}".format(position_id,len(position_
    # display all historical orders having a specified position ticket
    for position_order in position_history_orders:
        print(position_order)
    print()
    # display these orders as a table using pandas.DataFrame
    df=pd.DataFrame(list(position_history_orders),columns=position_history_orders[0]._
    df.drop(['time_expiration','type_time','state','position_by_id','reason','volume_
    df['time_setup'] = pd.to_datetime(df['time_setup'], unit='s')
    df['time_done'] = pd.to_datetime(df['time_done'], unit='s')
    print(df)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

history_orders_get(2020-01-01 00:00:00, 2020-03-25 17:17:32.058795, group="*GBP*")=14

Total history orders on position #530218319: 2
TradeOrder(ticket=530218319, time_setup=1582282114, time_setup_msc=1582282114681, time
TradeOrder(ticket=535548147, time_setup=1583176242, time_setup_msc=1583176242265, time

      ticket          time_setup  time_setup_msc           time_done  time_done_msc  t
0  530218319 2020-02-21 10:48:34   1582282114681 2020-02-21 16:49:37  1582303777582
1  535548147 2020-03-02 19:10:42   1583176242265 2020-03-02 19:10:42  1583176242265
```

See also

history_deals_total, history_deals_get

history_deals_total

Get the number of deals in trading history within the specified interval.

```
history_deals_total(
   date_from,     // date the deals are requested from
   date_to        // date, up to which the deals are requested
   )
```

Parameters

date_from

[in] Date the deals are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

date_to

[in] Date, up to which the deals are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter.

Return Value

Integer value.

Note

The function is similar to HistoryDealsTotal.

Example:

```
from datetime import datetime
import MetaTrader5 as mt5
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)

# establish connection to MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get the number of deals in history
from_date=datetime(2020,1,1)
to_date=datetime.now()
deals=mt5.history_deals_total(from_date, to_date)
if deals>0:
    print("Total deals=",deals)
else:
    print("Deals not found in history")

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()
```

See also

history_deals_get, history_orders_total

history_deals_get

Get deals from trading history within the specified interval with the ability to filter by ticket or position.

Call specifying a time interval. Return all deals falling within the specified interval.

```
history_deals_get(
   date_from,                 // date the deals are requested from
   date_to,                   // date, up to which the deals are requested
   group="GROUP"          // filter for selecting deals for symbols
   )
```

Call specifying the order ticket. Return all deals having the specified order ticket in the DEAL_ORDER property.

```
history_deals_get(
   ticket=TICKET          // order ticket
)
```

Call specifying the position ticket. Return all deals having the specified position ticket in the DEAL_POSITION_ID property.

```
history_deals_get(
   position=POSITION    // position ticket
)
```

Parameters

date_from

[in] Date the orders are requested from. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter is specified first.

date_to

[in] Date, up to which the orders are requested. Set by the 'datetime' object or as a number of seconds elapsed since 1970.01.01. Required unnamed parameter is specified second.

group="GROUP"

[in] The filter for arranging a group of necessary symbols. Optional named parameter. If the group is specified, the function returns only deals meeting a specified criteria for a symbol name.

ticket=TICKET

[in] Ticket of an order (stored in DEAL_ORDER) all deals should be received for. Optional parameter. If not specified, the filter is not applied.

position=POSITION

[in] Ticket of a position (stored in DEAL_POSITION_ID) all deals should be received for. Optional parameter. If not specified, the filter is not applied.

Return Value

Return info in the form of a named tuple structure (namedtuple). Return None in case of an error. The info on the error can be obtained using last_error().

Note

The function allows receiving all history deals within a specified period in a single call similar to the HistoryDealsTotal and HistoryDealSelect tandem.

The *group* parameter allows sorting out deals by symbols. '*' can be used at the beginning and the end of a string.

The *group* parameter may contain several comma separated conditions. A condition can be set as a mask using '*'. The logical negation symbol '!' can be used for an exclusion. All conditions are applied sequentially, which means conditions of including to a group should be specified first followed by an exclusion condition. For example, *group*="*, !EUR" means that deals for all symbols should be selected first and the ones containing "EUR" in symbol names should be excluded afterwards.

Example:

```
import MetaTrader5 as mt5
from datetime import datetime
import pandas as pd
pd.set_option('display.max_columns', 500) # number of columns to be displayed
pd.set_option('display.width', 1500)      # max table width to display
# display data on the MetaTrader 5 package
print("MetaTrader5 package author: ",mt5.__author__)
print("MetaTrader5 package version: ",mt5.__version__)
print()
# establish connection to the MetaTrader 5 terminal
if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

# get the number of deals in history
from_date=datetime(2020,1,1)
to_date=datetime.now()
# get deals for symbols whose names contain "GBP" within a specified interval
deals=mt5.history_deals_get(from_date, to_date, group="*GBP*")
if deals==None:
    print("No deals with group=\"*USD*\", error code={}".format(mt5.last_error()))
elif len(deals)> 0:
    print("history_deals_get({}, {}, group=\"*GBP*\")={}".format(from_date,to_date,len

# get deals for symbols whose names contain neither "EUR" nor "GBP"
deals = mt5.history_deals_get(from_date, to_date, group="*,!*EUR*,!*GBP*")
if deals == None:
    print("No deals, error code={}".format(mt5.last_error()))
elif len(deals) > 0:
    print("history_deals_get(from_date, to_date, group=\"*,!*EUR*,!*GBP*\") =", len(de
    # display all obtained deals 'as is'
    for deal in deals:
        print("  ",deal)
    print()
    # display these deals as a table using pandas.DataFrame
    df=pd.DataFrame(list(deals),columns=deals[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    print(df)
print("")

# get all deals related to the position #530218319
position_id=530218319
position_deals = mt5.history_deals_get(position=position_id)
if position_deals == None:
    print("No deals with position #{}".format(position_id))
    print("error code =", mt5.last_error())
elif len(position_deals) > 0:
    print("Deals with position id #{}: {}".format(position_id, len(position_deals)))
    # display these deals as a table using pandas.DataFrame
    df=pd.DataFrame(list(position_deals),columns=position_deals[0]._asdict().keys())
    df['time'] = pd.to_datetime(df['time'], unit='s')
    print(df)

# shut down connection to the MetaTrader 5 terminal
mt5.shutdown()

Result:
MetaTrader5 package author:  MetaQuotes Software Corp.
MetaTrader5 package version:  5.0.29

history_deals_get(from_date, to_date, group="*GBP*") = 14

history_deals_get(from_date, to_date, group="*,!*EUR*,!*GBP*") = 7
   TradeDeal(ticket=506966741, order=0, time=1582202125, time_msc=1582202125419, type=
   TradeDeal(ticket=507962919, order=530218319, time=1582303777, time_msc=158230377758
   TradeDeal(ticket=513149059, order=535548147, time=1583176242, time_msc=158317624226
   TradeDeal(ticket=516943494, order=539349382, time=1583510003, time_msc=158351000389
   TradeDeal(ticket=516943915, order=539349802, time=1583510025, time_msc=158351002505
   TradeDeal(ticket=517139682, order=539557870, time=1583520201, time_msc=158352020122
   TradeDeal(ticket=517139716, order=539557909, time=1583520202, time_msc=158352020297

      ticket      order                time       time_msc  type  entry  magic  positi
0  506966741          0 2020-02-20 12:35:25  1582202125419     2      0      0
1  507962919  530218319 2020-02-21 16:49:37  1582303777582     0      0      0    5302
2  513149059  535548147 2020-03-02 19:10:42  1583176242265     1      1      0    5302
3  516943494  539349382 2020-03-06 15:53:23  1583510003895     1      0      0    5393
4  516943915  539349802 2020-03-06 15:53:45  1583510025054     0      0      0    5393
5  517139682  539557870 2020-03-06 18:43:21  1583520201227     0      1      0    5393
6  517139716  539557909 2020-03-06 18:43:22  1583520202971     1      1      0    5393

Deals with position id #530218319: 2
      ticket      order                time       time_msc  type  entry  magic  positi
0  507962919  530218319 2020-02-21 16:49:37  1582303777582     0      0      0    5302
1  513149059  535548147 2020-03-02 19:10:42  1583176242265     1      1      0    5302
```

See also

history_deals_total, history_orders_get

ONNX Models in Machine Learning

ONNX (Open Neural Network Exchange) is an open-source format for machine learning models. This project has several major advantages:

- ONNX is supported by large companies such as Microsoft, Facebook, Amazon and other partners.
- Its open format enables format conversions between different machine learning toolkits, while Microsoft's ONNXMLTools allows converting models to the ONNX format.
- MQL5 provides automatic data type conversion for model inputs and outputs if the passed parameter type does not match the model.
- ONNX models can be created using various machine learning tools. They are currently supported in Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch and OpenCV. Interfaces for other popular frameworks and libraries are also available.
- With the MQL5 language, you can implement an ONNX model in a trading strategy and use it along with all the advantages of the MetaTrader 5 platform for efficient operations in the financial markets.
- Before tunning a model for live trading, you can test the model behavior on historical data in the Strategy Tester, without using third-party tools.

MQL5 provides the following functions for working with ONNX:

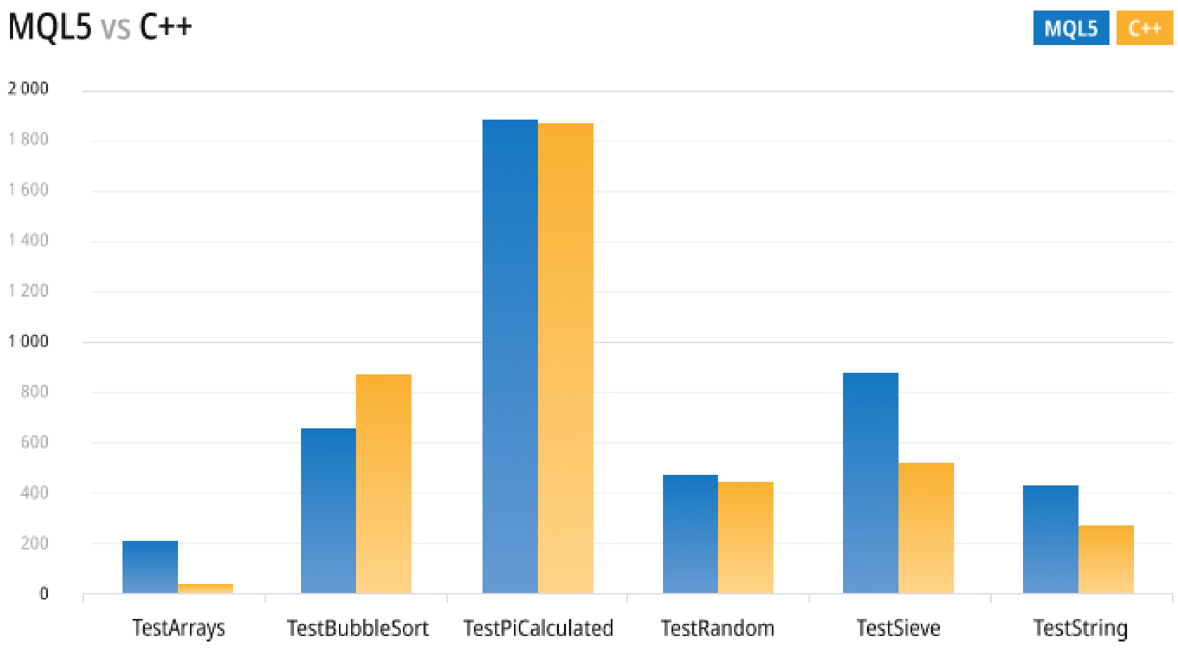
Function	Action
OnnxCreate	Create an ONNX session, loading a model from an *.onnx file
OnnxCreateFromBuffer	Create an ONNX session, loading a model from a data array
OnnxRelease	Close an ONNX session
OnnxRun	Run an ONNX model
OnnxGetInputCount	Get the number of inputs in an ONNX model
OnnxGetOutputCount	Get the number of outputs in an ONNX model
OnnxGetInputName	Get the name of a model's input by index
OnnxGetOutputName	Get the name of a model's output by index
OnnxGetInputTypeInfo	Get the description of the input type from the model
OnnxGetOutputTypeInfo	Get the description of the output type from the model
OnnxSetInputShape	Set the shape of a model's input data by index
OnnxSetOutputShape	Set the shape of a model's output data by index

ONNX Support in MQL5

ONNX is an open format build to represent machine learning models. This standard defines a common set of operators and a common file format to enable developers to use models with different frameworks, tools, runtimes and compilers.

Thus, the open ONNX format allows you to receive and transfer machine learning models between different platforms and machine learning toolkits. The ONNX is implemented in the MQL5 language to enable AI developers to run created models in the high-performance execution environment provided by the MetaTrader 5 platform.

The MQL5 execution speed is comparable to that of C++ applications. This is proved by the execution results of standard tests on MQL5 and C++. The lower the bar, the less time (in milliseconds) spent on execution and the better the result. The tests have been conducted on Windows 10 (build 17763) x64, Xeon E5-2630 v4 @ 2.20GHz, Memory: 65457 Mb.



The new asynchronous trading operations and the native ONNX support provide you with new opportunities which were previously available only to a handful of professional AI developers and institutional traders. ONNX support in MQL5 enables traders to train models for financial market trading in their preferred development environment and then to trade with low network costs, high order book update speeds and asynchronous order submission.

Currently, ONNX is being developed and maintained by partner companies such as Microsoft, Facebook, Amazon and others, which guarantees the further development of this open project.

Format Conversion

ONNX is an open format, which allows using models from different machine learning toolkits. This format is supported by many frameworks, including Chainer, Caffee2 and PyTorch.

One of the most popular tools for converting models to the ONNX format is Microsoft's ONNXMLTools.

ONNXMLTools installation and use instructions are available at the GitHub repo. The following toolkits are currently supported:

- Keras (a wrapper of keras2onnx converter)
- Tensorflow (a wrapper of tf2onnx converter)
- scikit-learn (a wrapper of skl2onnx converter)
- Apple Core ML
- Spark ML (experimental)
- LightGBM
- libscm;
- XGBoost;
- H2O
- CatBoost

ONNXMLTools can be easily installed. For installation details and model conversion examples, please see the project page at https://github.com/onnx/onnxmltools#install.

Autoconvert input and output values when running ONNX models

The current ONNX version in MQL5 supports only tensors for input/output values. Tensors are data arrays with the elements of the following data types:

ONNX type	Corresponds to MQL5 type
ONNX_DATA_TYPE_BOOL	bool
ONNX_DATA_TYPE_FLOAT	float
ONNX_DATA_TYPE_UINT8	uchar
ONNX_DATA_TYPE_INT8	char
ONNX_DATA_TYPE_UINT16	ushort
ONNX_DATA_TYPE_INT16	short
ONNX_DATA_TYPE_INT32	int
ONNX_DATA_TYPE_INT64	long
ONNX_DATA_TYPE_FLOAT16	—
ONNX_DATA_TYPE_DOUBLE	double
ONNX_DATA_TYPE_UINT32	uint
ONNX_DATA_TYPE_UINT64	ulong
ONNX_DATA_TYPE_COMPLEX64	—
ONNX_DATA_TYPE_COMPLEX128	complex
ONNX_DATA_TYPE_BFLOAT16	—
ONNX_DATA_TYPE_STRING	—

Only arrays, vectors and matrices (we will refer to them as the **Data**) can be fed into ONNX models as input/output values.

If the parameter types does not match the ONNX model's parameter type, and the OnnxRun is called without the ONNX_NO_CONVERSION flag specified, automatic data conversion will be applied. Autoconversion implies that before running an ONNX model, user **Data** will be copied into ONNX tensors with the relevant conversion.

When an ONNX model is run without the autoconversion, the model will be calculated using the **Data** without any additional copying.

> **IMPORTANT!** Autoconversion does not control overflow (truncate), therefore you should carefully monitor the data and the data types input into the ONNX model.

Autoconversion supports the following ONNX types:

- ONNX_DATA_TYPE_BOOL
- ONNX_DATA_TYPE_FLOAT
- ONNX_DATA_TYPE_UINT8
- ONNX_DATA_TYPE_INT8
- ONNX_DATA_TYPE_UINT16
- ONNX_DATA_TYPE_INT16
- ONNX_DATA_TYPE_INT32
- ONNX_DATA_TYPE_INT64
- ONNX_DATA_TYPE_FLOAT16
- ONNX_DATA_TYPE_DOUBLE
- ONNX_DATA_TYPE_UINT32
- ONNX_DATA_TYPE_UINT64
- ONNX_DATA_TYPE_COMPLEX64
- ONNX_DATA_TYPE_COMPLEX128

Unsupported types:

- ONNX_DATA_TYPE_BFLOAT16
- ONNX_DATA_TYPE_STRING

Autoconversion Rules by Tensor Types

If the MQL5 type is not included into the list of types supported by the model, running the ONNX model will return the ERR_ONNX_NOT_SUPPORTED error (error code 5802).

Note: During autoconversion, the color type is processed as uint, while datetime is processed as long.

Autoconversion of input values

ONNX type (tensor item type)	MQL5 type supported by autoconversion
ONNX_DATA_TYPE_BOOL	bool, char, uchar, short, ushort, int, color, uint, datetime, long, folat, double, complex During conversion, **Data** elements are checked by a simple comparison against 0
ONNX_DATA_TYPE_FLOAT16	float, double
ONNX_DATA_TYPE_FLOAT	char, uchar, short, ushort, int, color, uint, datetime, long, ulong, float, double
ONNX_DATA_TYPE_UINT8	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT8	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT16	See ONNX_DATA_TYPE_FLOAT

ONNX type (tensor item type)	MQL5 type supported by autoconversion
ONNX_DATA_TYPE_INT16	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT32	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT64	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_DOUBLE	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT32	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT64	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_COMPLEX64	complex
ONNX_DATA_TYPE_COMPLEX128	complex

Autoconversion of output values

ONNX type (tensor item type)	MQL5 type supported by autoconversion
ONNX_DATA_TYPE_BOOL	bool, char, uchar, short, ushort, int, color, uint, datetime, long, folat, double, complex If the tensor element is zero, then the Data element is set to 0; otherwise, the value is 1
ONNX_DATA_TYPE_FLOAT16	float, double
ONNX_DATA_TYPE_FLOAT	char, uchar, short, ushort, int, color, uint, datetime, long, ulong, float, double
ONNX_DATA_TYPE_UINT8	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT8	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT16	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT16	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT32	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_INT64	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_DOUBLE	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT32	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_UINT64	See ONNX_DATA_TYPE_FLOAT
ONNX_DATA_TYPE_COMPLEX64	complex
ONNX_DATA_TYPE_COMPLEX128	complex

See also

Type Casting

Creating a Model

Multiple methods are available to obtain a ready model in the ONNX format. The popular ONNX Model Zoo library contains several pre-trained ONNX models for different types of tasks. The advantage of this collection is that each model's notebook contains links to the training dataset and references to the original paper that describes the model architecture.

Most machine learning frameworks use Python. To install the ONNX runtime for Python, use one of the following commands:

```
pip install onnxruntime       # CPU build
pip install onnxruntime-gpu   # GPU build
```

To invoke the ONNX runtime in Python, use the following command

```
import onnxruntime
session = onnxruntime.InferenceSession("path to model")
```

For model inputs and outputs, check out the relevant model's documentation. You can also use visualization tools to view the model, such as Netron or WinML Dashboard. In the ONNX runtime, you can also query a model's metadata and its inputs and outputs:

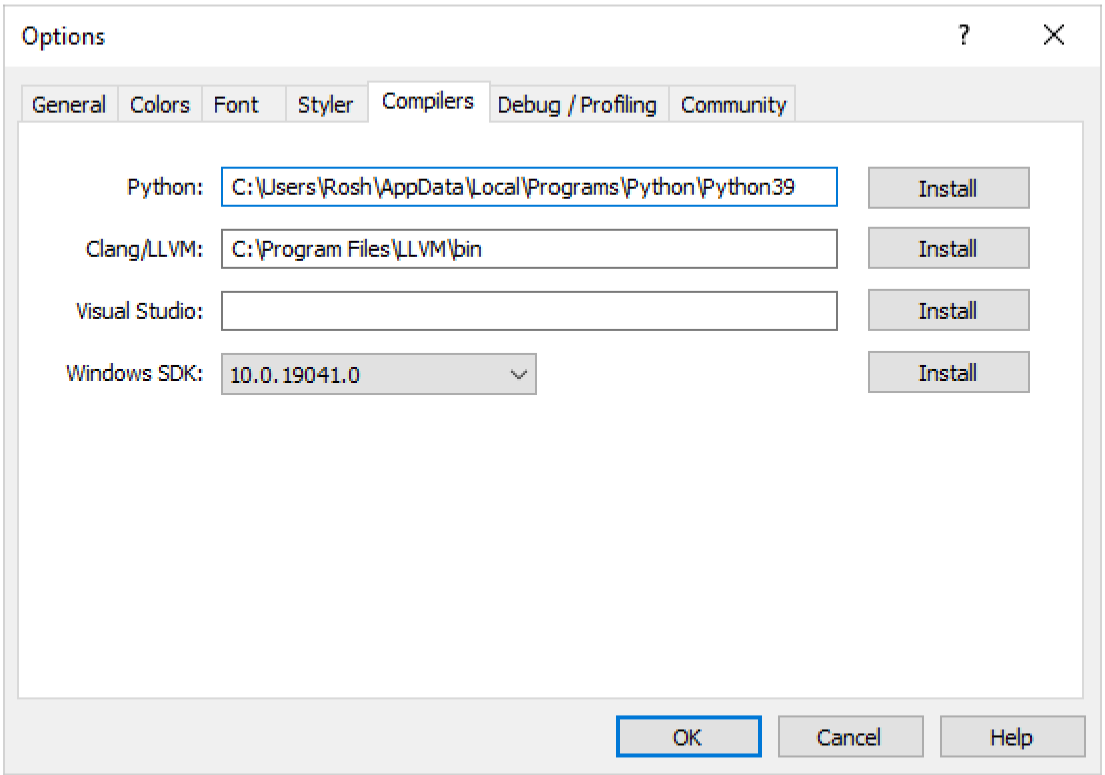
```
results = session.run(["output1", "output2"], {
                      "input1": indata1, "input2": indata2})
results = session.run([], {"input1": indata1, "input2": indata2})
```

You can create ONNX models directly in the MetaTrader 5 terminal or in MetaEditor using Python.

Python in MetaTrader 5

The MetaTrader 5 provides out-of-the-box support for Python scripts. To enable these operations, the terminal developers provide MetaTrader5 module for Python: https://pypi.org/project/MetaTrader5.

In the MetaEditor integrated development environment, in addition to creating applications in MQL5, you can also run Python scripts directly from the editor. To do this, specify the path to the executable in MetaEditor settings:



Python: C:\Users\Rosh\AppData\Local\Programs\Python\Python39
Clang/LLVM: C:\Program Files\LLVM\bin
Visual Studio:
Windows SDK: 10.0.19041.0

If Python is not installed on your computer, click Install to download the installation file.

You can create a Python script in MetaEdtior or upload it to the terminal's data folder and immediately run it using the F7 (Compile) key. This will open the MetaTrader 5 terminal with the script running on the current chart. Messages from the Python console (stdout, stderr) will be displayed under the Errors section.

Operations with models in MetaTrader 5

The MQL5 language allows you to run ONNX models directly in the MetaTrader 5 terminal. This is done in three steps:

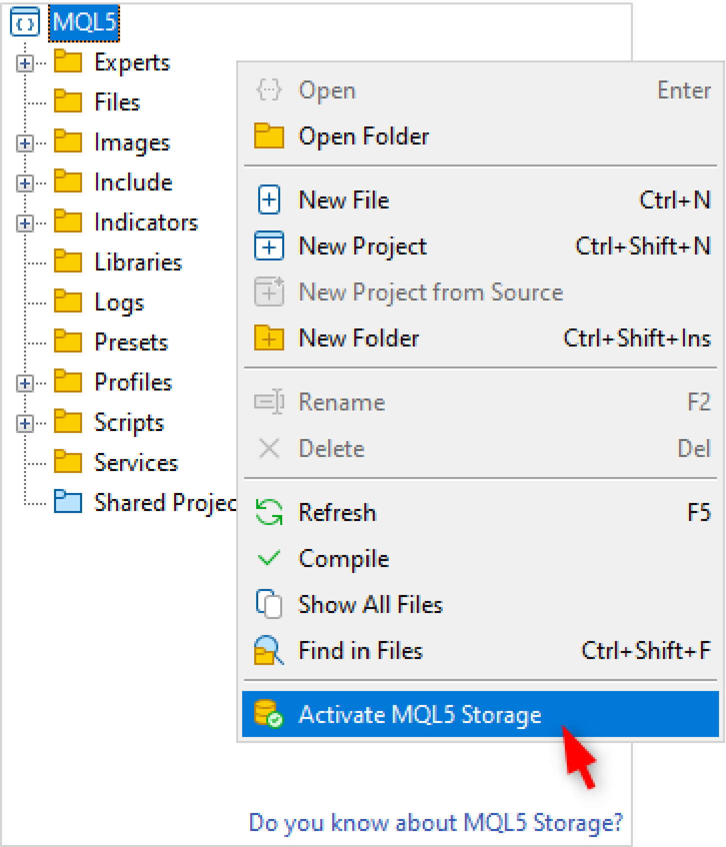
1. Train the model in a third-party platform, such as Python.
2. Convert the model to ONNX.
3. Include the ONNX model into an Expert Advisor using ONNX function and run in the MetaTrader 5 terminal.

Python integration in MQL5 allows running a python script and saving an ONNX model in the MetaEditor or run it directly on a chart in MetaTrader 5. You can train the model using a pre-written Python script as often as you need right in the terminal. The library includes ready-made functions for obtaining price data, which can be input into an ONNX model:

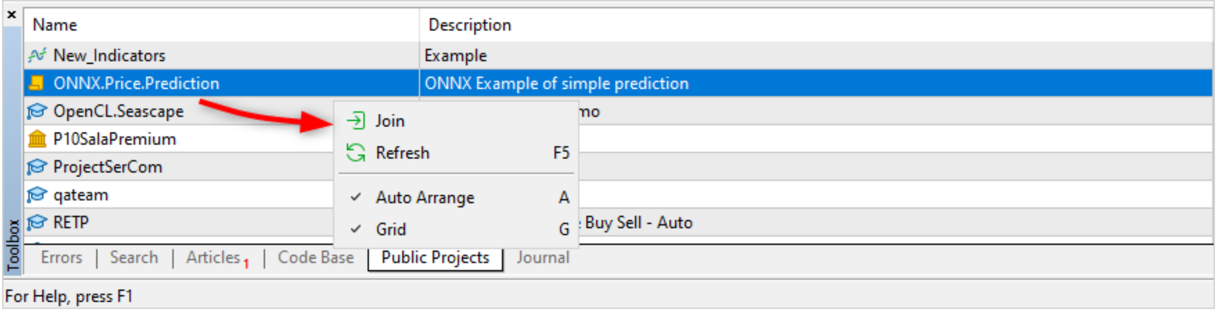
- copy_rates_from - get bars starting from the specified date
- copy_rates_from_pos - get bars starting from the specified index
- copy_rates_range - get bars for the specified date range
- copy_ticks_from - get ticks starting from the specified date
- copy_ticks_range - get ticks for the specified date range

Model example

En example of a finished ONNX model is available in public projects. You should first activate MQL5 Storage in the Navigator by specifying you MQL5 login in MetaEditor settings (case sensitive).

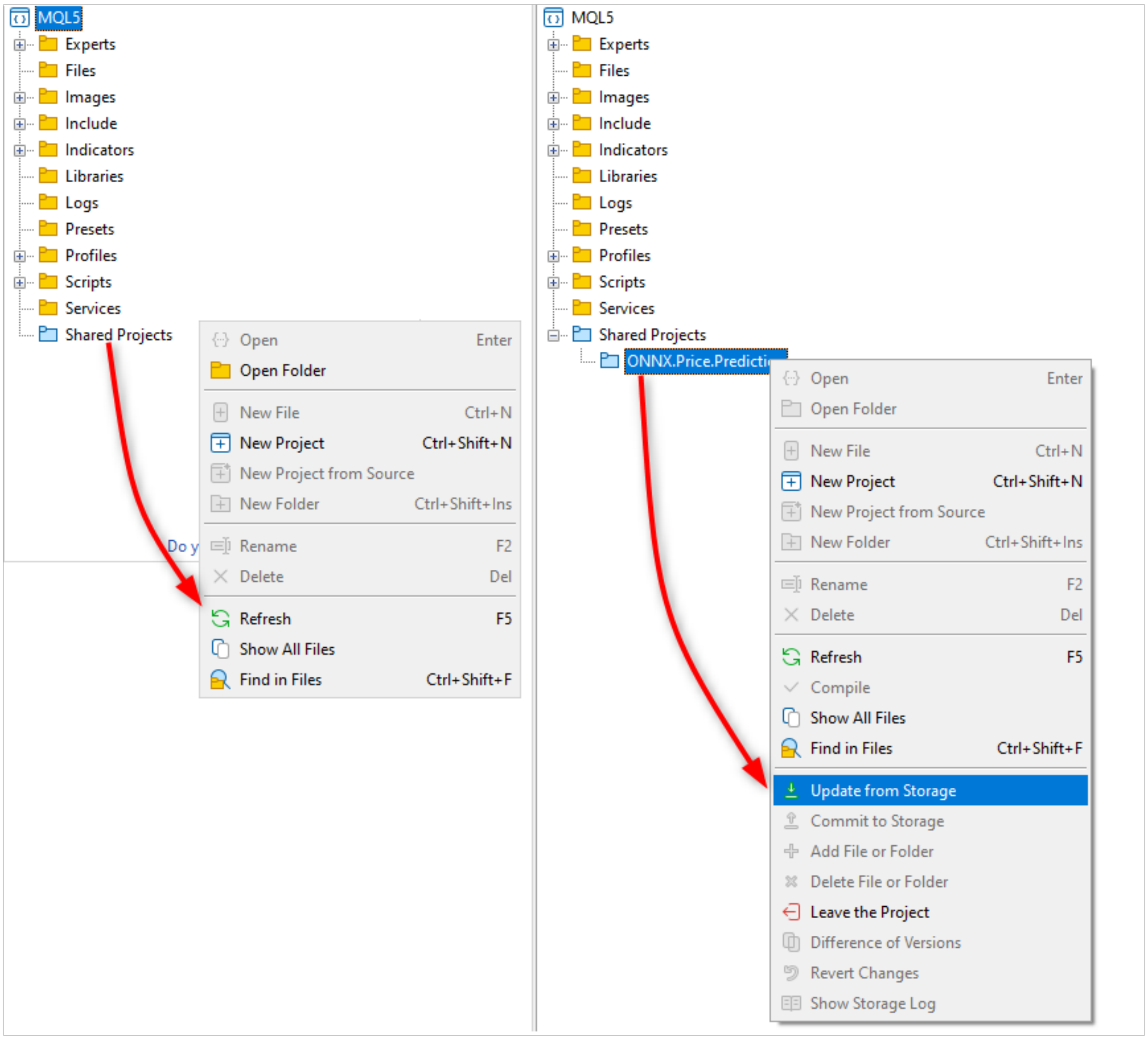

MQL5
Experts
Files
Images
Include
Indicators
Libraries
Logs
Presets
Profiles
Scripts
Services
Shared Projec
Open Enter
Open Folder
New File Ctrl+N
New Project Ctrl+Shift+N
New Project from Source
New Folder Ctrl+Shift+Ins
Rename F2
Delete Del
Refresh F5
Compile
Show All Files
Find in Files Ctrl+Shift+F
Activate MQL5 Storage
Do you know about MQL5 Storage?

After activation, find the ONNX.Price.Prediction project and join it via the context menu command.

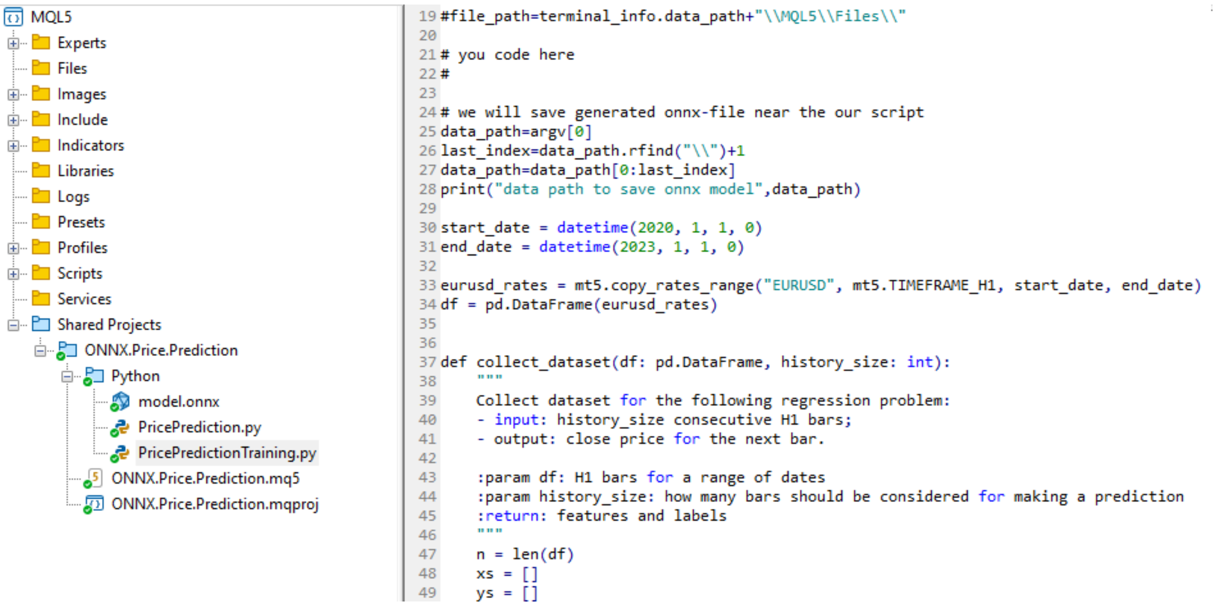

Name	Description
New_Indicators	Example
ONNX.Price.Prediction	ONNX Example of simple prediction
OpenCL.Seascape	mo
P10SalaPremium	
ProjectSerCom	
qateam	
RETP	: Buy Sell - Auto

Join
Refresh F5
Auto Arrange A
Grid G
Toolbox
Errors | Search | Articles 1 | Code Base | Public Projects | Journal
For Help, press F1

Next, update the project from MQL5 Storage.


MQL5
Experts
Files
Images
Include
Indicators
Libraries
Logs
Presets
Profiles
Scripts
Services
Shared Projects
Do y
Open Enter
Open Folder
New File Ctrl+N
New Project Ctrl+Shift+N
New Project from Source
New Folder Ctrl+Shift+Ins
Rename F2
Delete Del
Refresh F5
Show All Files
Find in Files Ctrl+Shift+F
MQL5
Experts
Files
Images
Include
Indicators
Libraries
Logs
Presets
Profiles
Scripts
Services
Shared Projects
ONNX.Price.Predicti
Open Enter
Open Folder
New File Ctrl+N
New Project Ctrl+Shift+N
New Project from Source
New Folder Ctrl+Shift+Ins
Rename F2
Delete Del
Refresh F5
Compile
Show All Files
Find in Files Ctrl+Shift+F
Update from Storage
Commit to Storage
Add File or Folder
Delete File or Folder
Leave the Project
Difference of Versions
Revert Changes
Show Storage Log

The project contains an ONNX model, two python scripts, an MQL5 script for project operation, and an MQL5 project file (ONNX.Price.Prediction.mqproj).

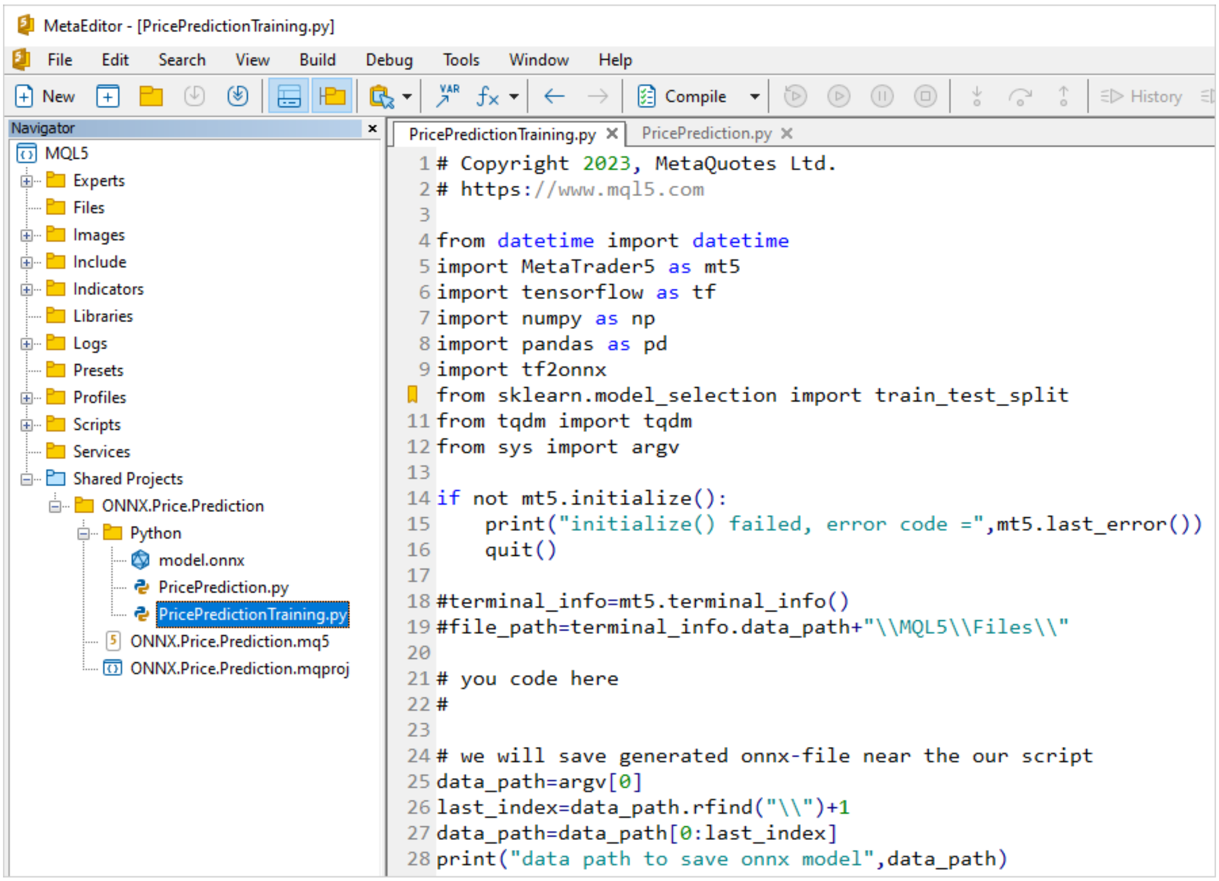

MQL5
Experts
Files
Images
Include
Indicators
Libraries
Logs
Presets
Profiles
Scripts
Services
Shared Projects
ONNX.Price.Prediction
Python
model.onnx
PricePrediction.py
PricePredictionTraining.py
ONNX.Price.Prediction.mq5
ONNX.Price.Prediction.mqproj

```
#file_path=terminal_info.data_path+"\\MQL5\\Files\\"

# you code here
#

# we will save generated onnx-file near the our script
data_path=argv[0]
last_index=data_path.rfind("\\")+1
data_path=data_path[0:last_index]
print("data path to save onnx model",data_path)

start_date = datetime(2020, 1, 1, 0)
end_date = datetime(2023, 1, 1, 0)

eurusd_rates = mt5.copy_rates_range("EURUSD", mt5.TIMEFRAME_H1, start_date, end_date)
df = pd.DataFrame(eurusd_rates)


def collect_dataset(df: pd.DataFrame, history_size: int):
    """
    Collect dataset for the following regression problem:
    - input: history_size consecutive H1 bars;
    - output: close price for the next bar.

    :param df: H1 bars for a range of dates
    :param history_size: how many bars should be considered for making a prediction
    :return: features and labels
    """
    n = len(df)
    xs = []
    ys = []
```

You can create the ONNX model yourself using the PricePredictionTraining.py script included in the project. To do this, you should first install the required modules from the command line.

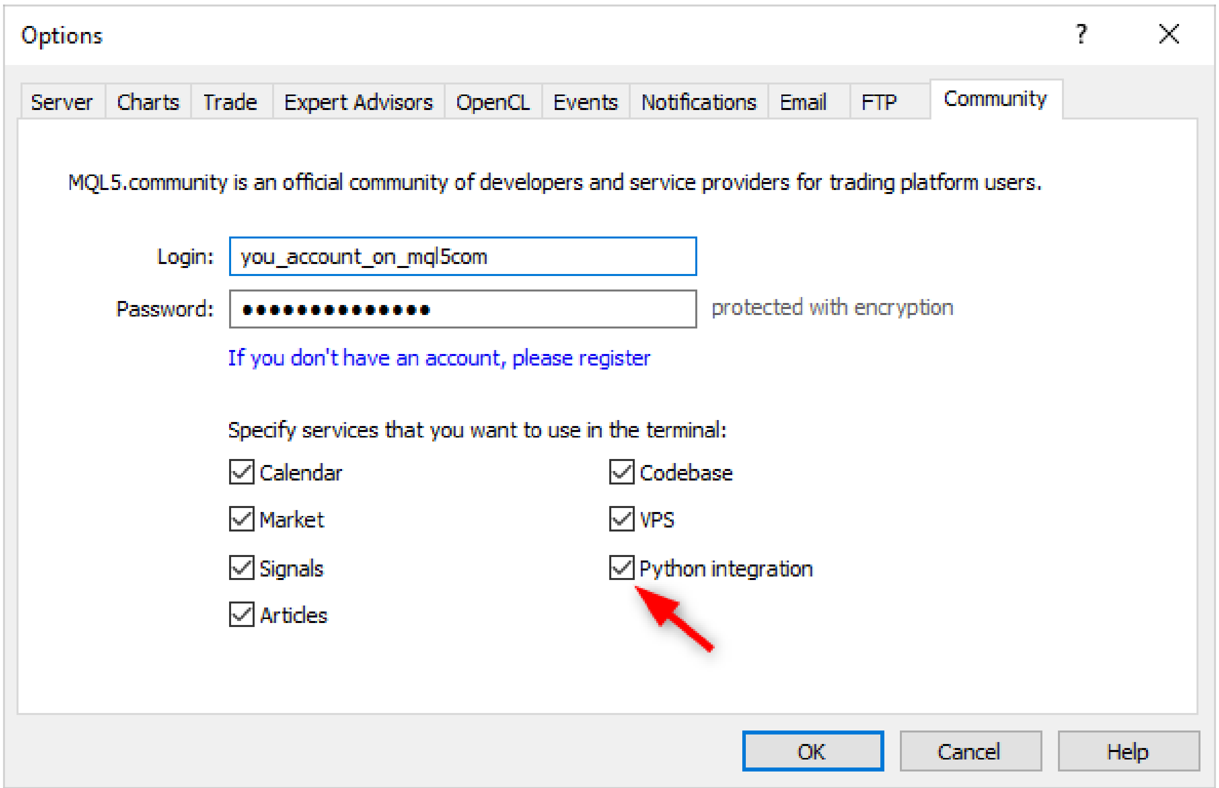
```
python.exe -m pip install --upgrade pip
python -m pip install --upgrade tensorflow
python -m pip install --upgrade pandas
python -m pip install --upgrade scikit-learn
python -m pip install --upgrade matplotlib
python -m pip install --upgrade tqdm
python -m pip install --upgrade metatrader5
python -m pip install --upgrade onnx==1.12
python -m pip install --upgrade tf2onnx
python -m pip install --upgrade numpy
python -m pip install onnxruntime
```

After installing the modules, open the PricePredictionTraining.py script in MetaEditor and run it with the Compile button or with the F7 key.


MetaEditor - [PricePredictionTraining.py]
File
Edit
Search
View
Build
Debug
Tools
Window
Help
New
Compile
History
Navigator
MQL5
Experts
Files
Images
Include
Indicators
Libraries
Logs
Presets
Profiles
Scripts
Services
Shared Projects
ONNX.Price.Prediction
Python
model.onnx
PricePrediction.py
PricePredictionTraining.py
ONNX.Price.Prediction.mq5
ONNX.Price.Prediction.mqproj
PricePredictionTraining.py
PricePrediction.py

```python
# Copyright 2023, MetaQuotes Ltd.
# https://www.mql5.com

from datetime import datetime
import MetaTrader5 as mt5
import tensorflow as tf
import numpy as np
import pandas as pd
import tf2onnx
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from sys import argv

if not mt5.initialize():
    print("initialize() failed, error code =",mt5.last_error())
    quit()

#terminal_info=mt5.terminal_info()
#file_path=terminal_info.data_path+"\\MQL5\\Files\\"

# you code here
#

# we will save generated onnx-file near the our script
data_path=argv[0]
last_index=data_path.rfind("\\")+1
data_path=data_path[0:last_index]
print("data path to save onnx model",data_path)
```

Before running the Python script, make sure that the MetaTrader 5 terminal is connected to a server with the EURUSD symbol available. For example, connect to the MetaQuotes-Demo server and check "Integration with Python" in terminal settings.


Options
?
Server
Charts
Trade
Expert Advisors
OpenCL
Events
Notifications
Email
FTP
Community
OK
Cancel
Help

MQL5.community is an official community of developers and service providers for trading platform users.

Login: you_account_on_mql5com

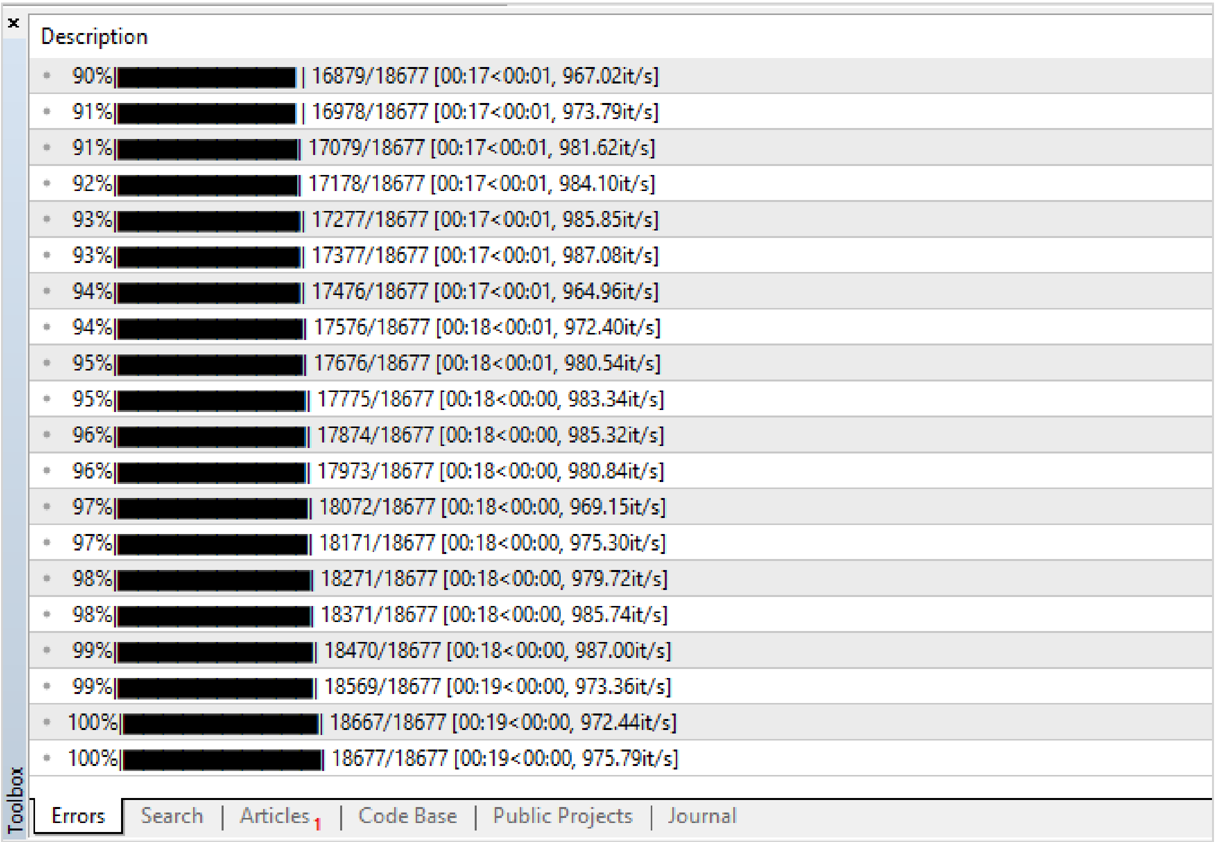
Password: ●●●●●●●●●●●●●● protected with encryption

If you don't have an account, please register

Specify services that you want to use in the terminal:

☑ Calendar
☑ Market
☑ Signals
☑ Articles
☑ Codebase
☑ VPS
☑ Python integration

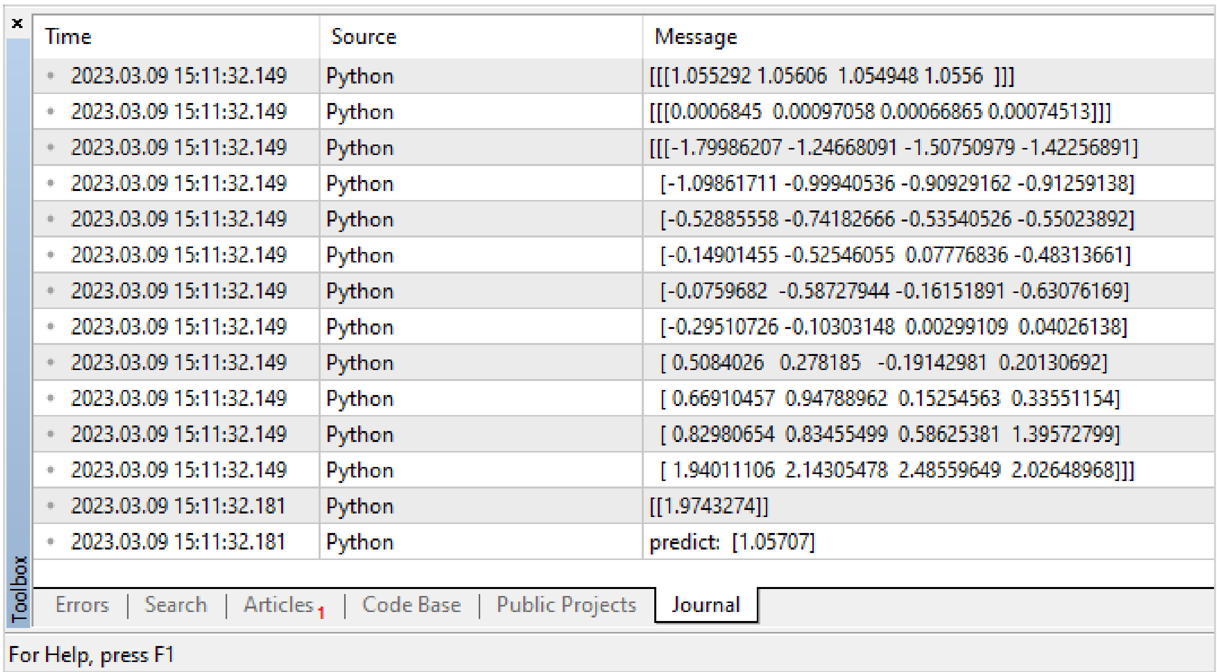
While training the network, MetaEditor will print messages from the Python script until the training is complete.



```
Description
 90%|█████████ | 16879/18677 [00:17<00:01, 967.02it/s]
 91%|█████████ | 16978/18677 [00:17<00:01, 973.79it/s]
 91%|█████████▏| 17079/18677 [00:17<00:01, 981.62it/s]
 92%|█████████▏| 17178/18677 [00:17<00:01, 984.10it/s]
 93%|█████████▎| 17277/18677 [00:17<00:01, 985.85it/s]
 93%|█████████▎| 17377/18677 [00:17<00:01, 987.08it/s]
 94%|█████████▎| 17476/18677 [00:17<00:01, 964.96it/s]
 94%|█████████▍| 17576/18677 [00:18<00:01, 972.40it/s]
 95%|█████████▍| 17676/18677 [00:18<00:01, 980.54it/s]
 95%|█████████▌| 17775/18677 [00:18<00:00, 983.34it/s]
 96%|█████████▌| 17874/18677 [00:18<00:00, 985.32it/s]
 96%|█████████▌| 17973/18677 [00:18<00:00, 980.84it/s]
 97%|█████████▋| 18072/18677 [00:18<00:00, 969.15it/s]
 97%|█████████▋| 18171/18677 [00:18<00:00, 975.30it/s]
 98%|█████████▊| 18271/18677 [00:18<00:00, 979.72it/s]
 98%|█████████▊| 18371/18677 [00:18<00:00, 985.74it/s]
 99%|█████████▉| 18470/18677 [00:18<00:00, 987.00it/s]
 99%|█████████▉| 18569/18677 [00:19<00:00, 973.36it/s]
100%|█████████▉| 18667/18677 [00:19<00:00, 972.44it/s]
100%|██████████| 18677/18677 [00:19<00:00, 975.79it/s]
```

When the result is 100%, the ONNX model is ready and it is saved to the project folder at <terminal data directory>\MQL5\Shared Projects\ONNX.Price.Prediction\Python.

You can check the resulting model by running the second script PricePrediction.py, pressing **F7**.



```
Time                     Source   Message
2023.03.09 15:11:32.149  Python   [[[1.055292 1.05606  1.054948 1.0556  ]]]
2023.03.09 15:11:32.149  Python   [[[0.0006845  0.00097058 0.00066865 0.00074513]]]
2023.03.09 15:11:32.149  Python   [[[-1.79986207 -1.24668091 -1.50750979 -1.42256891]
2023.03.09 15:11:32.149  Python     [-1.09861711 -0.99940536 -0.90929162 -0.91259138]
2023.03.09 15:11:32.149  Python     [-0.52885558 -0.74182666 -0.53540526 -0.55023892]
2023.03.09 15:11:32.149  Python     [-0.14901455 -0.52546055  0.07776836 -0.48313661]
2023.03.09 15:11:32.149  Python     [-0.0759682  -0.58727944 -0.16151891 -0.63076169]
2023.03.09 15:11:32.149  Python     [-0.29510726 -0.10303148  0.00299109  0.04026138]
2023.03.09 15:11:32.149  Python     [ 0.5084026   0.278185   -0.19142981  0.20130692]
2023.03.09 15:11:32.149  Python     [ 0.66910457  0.94788962  0.15254563  0.33551154]
2023.03.09 15:11:32.149  Python     [ 0.82980654  0.83455499  0.58625381  1.39572799]
2023.03.09 15:11:32.149  Python     [ 1.94011106  2.14305478  2.48559649  2.02648968]]]
2023.03.09 15:11:32.181  Python   [[1.9743274]]
2023.03.09 15:11:32.181  Python   predict:  [1.05707]
```

Running a model

To run an ONNX model in MQL5, complete 3 steps:

1. Load the model from an *.onnx file using the OnnxCreate function or from an array using OnnxCreateFromBuffer.
2. Specify input and output data shapes using OnnxSetInputShape and OnnxSetOutputShape functions.
3. Run the model using the OnnxRun function, passing to it the relevant input and output parameters.
4. When needed, you can terminate the model operation using the OnnxRelease function.

When creating an ONNX model, you should consider the existing limits and restrictions, which are described at https://github.com/microsoft/onnxruntime/blob/rel-1.14.0/docs/OperatorKernels.md

Some of the examples of such restrictions are shown below:

Operation	Supported data types
ReduceSum	tensor(double), tensor(float), tensor(int32), tensor(int64)
Mul	tensor(bfloat16), tensor(double), tensor(float), tensor(float16), tensor(int32), tensor(int64), tensor(uint32), tensor(uint64)

Below is an MQL5 code example from the public project ONNX.Price.Prediction.

```
const long   ExtOutputShape[] = {1,1};    // model's output shape
const long   ExtInputShape [] = {1,10,4}; // model's input shape
#resource "Python/model.onnx" as uchar ExtModel[]// model as a resource
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   matrix rates;
//--- get 10 bars
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,2,10))
      return(-1);
//--- input a set of OHLC vectors
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- fill in the normalization matrices
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- normalize the input data
```

```
   x_norm-=mm;
   x_norm/=ms;
//--- create the model
   long handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- specify the shape of the input data
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- specify the shape of the output data
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("OnnxSetOutputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- convert normalized input data to float type
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- get the output data of the model here, i.e. the price prediction
   vectorf y_norm(1);
//--- run the model
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- print the output value of the model to the log
   Print(y_norm);
//--- do the reverse transformation to get the predicted price
   double y_pred=y_norm[0]*s[3]+m[3];
   Print("price predicted:",y_pred);
//--- complete operation
   OnnxRelease(handle);
   return(0);
  }
```

Script run example:

```
ONNX: Creating and using per session threadpools since use_per_session_threads_ is tru
ONNX: Dynamic block base set to 0
ONNX: Initializing session.
ONNX: Adding default CPU execution provider.
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Total shared scalar initializer count: 0
```

```
ONNX: Total fused reshape node count: 0
ONNX: Use DeviceBasedPartition as default
ONNX: Saving initialized tensors.
ONNX: Done saving initialized tensors
ONNX: Session successfully initialized.
[0.28188983]
predicted 1.0559258806393044
```

The MetaTrader 5 terminal has selected the optimal executor for calculations — ONNX Runtime Execution Provider. In this example, the model was executed on the CPU.

Let's modify the script to calculate the percentage of successful Close price predictions made based on the values of the preceding 10 bars.

```
#resource "Python/model.onnx" as uchar ExtModel[]// model as a resource

#define TESTS 10000  // number of test datasets
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart()
  {
//--- create the model
   long session_handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
   if(session_handle==INVALID_HANDLE)
     {
      Print("Cannot create model. Error ",GetLastError());
      return(-1);
     }

//--- since the input tensor size is not defined for the model, specify it explicitly
//--- first index is batch size, second index is series size, third index is number of
   const long input_shape[]={1,10,4};
   if(!OnnxSetInputShape(session_handle,0,input_shape))
     {
      Print("OnnxSetInputShape error ",GetLastError());
      return(-2);
     }

//--- since the output tensor size is not defined for the model, specify it explicitly
//--- first index is batch size, must match the batch size in the input tensor
//--- second index is number of predicted prices (only Close is predicted here)
   const long output_shape[]={1,1};
   if(!OnnxSetOutputShape(session_handle,0,output_shape))
     {
      Print("OnnxSetOutputShape error ",GetLastError());
      return(-3);
     }
//--- run tests
   vector closes(TESTS);      // vector to store validation prices
   vector predicts(TESTS);    // vector to store obtained predictions
   vector prev_closes(TESTS); // vector to store preceding prices

   matrix rates;              // matrix to get the OHLC series
   matrix splitted[2];        // two submatrices to divide the series into test and va
   ulong  parts[]={10,1};     // sizes of divided submatrices

//--- start from the previous bar
   for(int i=1; i<=TESTS; i++)
     {
      //--- get 11 bars
      rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,i,11);
      //--- divide the matrix into test and validation
      rates.Vsplit(parts,splitted);
      //--- take the Close price from the validation matrix
      closes[i-1]=splitted[1][3][0];
      //--- last Close in the tested series
      prev_closes[i-1]=splitted[0][3][9];

      //--- submit the test matrix of 10 bars to testing
      predicts[i-1]=PricePredictionTest(session_handle,splitted[0]);
      //--- runtime error
      if(predicts[i-1]<=0)
        {
         OnnxRelease(session_handle);
         return(-4);
        }
     }
//--- complete operation
   OnnxRelease(session_handle);
//--- evaluate if price movement was predicted correctly
   int    right_directions=0;
   vector delta_predicts=prev_closes-predicts;
   vector delta_actuals=prev_closes-closes;

   for(int i=0; i<TESTS; i++)
      if((delta_predicts[i]>0 && delta_actuals[i]>0) || (delta_predicts[i]<0 && delta_
         right_directions++;
   PrintFormat("right direction predictions = %.2f%%",(right_directions*100.0)/double
//---
   return(0);
  }
//+------------------------------------------------------------------+
//|  Prepare the data and run the model                              |
//+------------------------------------------------------------------+
double PricePredictionTest(const long session_handle,matrix& rates)
  {
   static matrixf input_data(10,4); // matrix for the transformed input
   static vectorf output_data(1);   // vector to receive the result
   static matrix mm(10,4);          // matrix of horizontal vectors Mean
   static matrix ms(10,4);          // matrix of horizontal vectors Std

//--- a set of OHLC vertical vectors must be input into the model
   matrix x_norm=rates.Transpose();
//--- normalize prices
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
   x_norm-=mm;
   x_norm/=ms;

//--- run the model
   input_data.Assign(x_norm);
   if(!OnnxRun(session_handle,ONNX_DEBUG_LOGS,input_data,output_data))
     {
      Print("OnnxRun error ",GetLastError());
      return(0);
     }
//--- unnormalize the price from the output value
   double y_pred=output_data[0]*s[3]+m[3];

   return(y_pred);
  }
```

Run the script: the prediction accuracy is about 51%

```
ONNX: Creating and using per session threadpools since use_per_session_threads_ is tru
ONNX: Dynamic block base set to 0
ONNX: Initializing session.
ONNX: Adding default CPU execution provider.
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Total shared scalar initializer count: 0
ONNX: Total fused reshape node count: 0
ONNX: Use DeviceBasedPartition as default
ONNX: Saving initialized tensors.
ONNX: Done saving initialized tensors
ONNX: Session successfully initialized.
right direction predictions = 51.34 %
```

Model validation in the Strategy Tester

Models created for operations in the financial markets can be validated in the MetaTrader 5 terminal Strategy Tester. This is the fastest and most convenient option, which eliminates the need to additionally emulate the market environment and trading conditions.

To test the model, let us create an Expert Advisor based on the code from the public project ONNX.Price.Prediction. This will require some edits.

Move the model creation to the OnInit function. The onnx session will be closed in OnDeinit. Locate the main model operations block to the OnTick handler.

Also, add obtaining of the Close price of the previous two bars, which is needed to compare the actual Close price and the prediction.

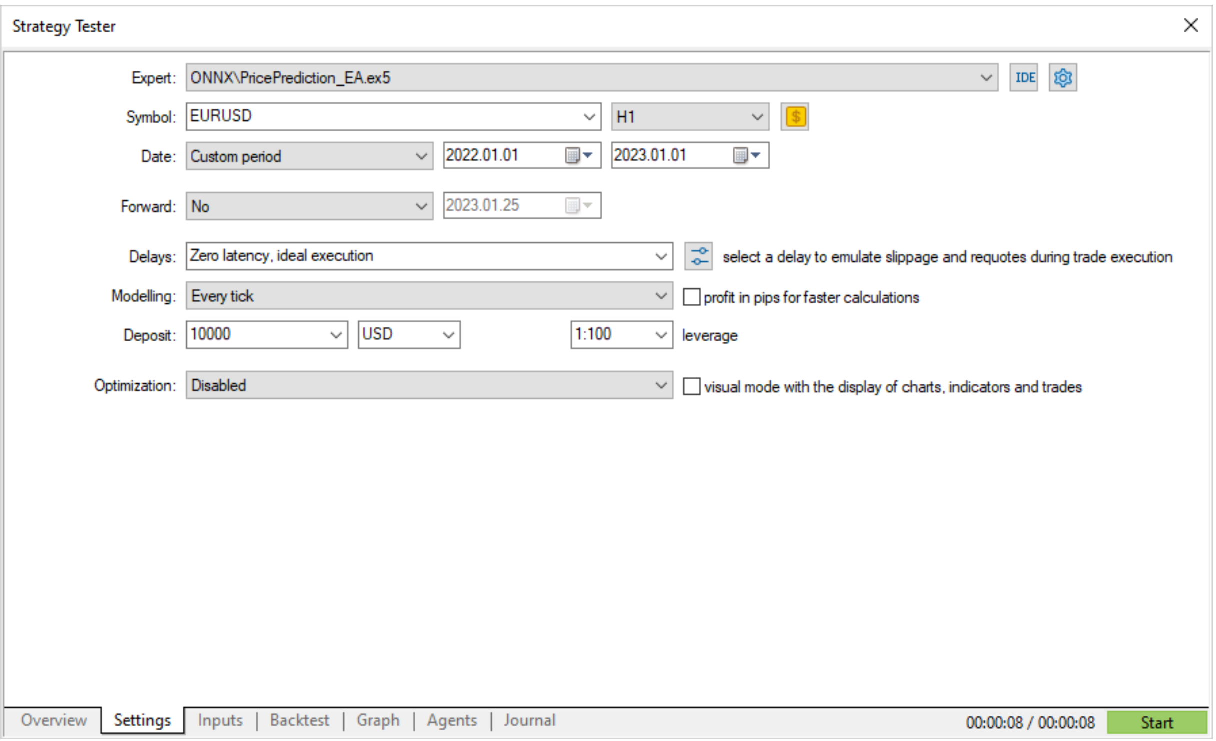
The Expert Advisor code is small and easy to read.

```
const long   ExtInputShape [] = {1,10,4}; // model's input shape
const long   ExtOutputShape[] = {1,1};    // model's output shape
#resource "Python/model.onnx" as uchar ExtModel[];// model as a resource

long handle;         // model handle
ulong predictions=0; // predictions counter
ulong confirmed=0;   // successful predictions counter
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- basic checks
   if(_Symbol!="EURUSD")
     {
      Print("Symbol must be EURUSD, testing aborted");
      return(-1);
     }
   if(_Period!=PERIOD_H1)
     {
      Print("Timeframe must be H1, testing aborted");
      return(-1);
     }
//--- create the model
   handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- specify the shape of the input data
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
```

```
//--- specify the shape of the output data
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("OnnxSetOutputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- complete model operation
   OnnxRelease(handle);
//--- calculate and output prediction statistics
   PrintFormat("Successfull predictions = %.2f %%",confirmed*100./double(predictions))
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   static datetime open_time=0;
   static double predict;
//--- check the current bar opening time
   datetime time=iTime(_Symbol,_Period,0);
   if(time==0)
     {
      PrintFormat("Failed to get Time(0), error %d", GetLastError());
      return;
     }
//--- if the opening time has not changed, exit until the next OnTick call
   if(time==open_time)
      return;
//--- get the Close prices of the last two completed bars
   double close[];
   int recieved=CopyClose(_Symbol,_Period,1,2,close);
   if(recieved!=2)
     {
      PrintFormat("CopyClose(2 bars) failed, error %d",GetLastError());
      return;
     }
   double delta_predict=predict-close[0]; // predicted price change
   double delta_actual=close[1]-close[0]; // actual price change
   if((delta_predict>0 && delta_actual>0) || (delta_predict<0 && delta_actual<0))
      confirmed++;
```

```
//--- calculate the Close price on the new bar to validate the price on the next bar
   matrix rates;
//--- get 10 bars
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,1,10))
      return;
//--- input a set of OHLC vectors
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- fill in the normalization matrices
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- normalize the input data
   x_norm-=mm;
   x_norm/=ms;
//--- convert normalized input data to float type
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- get the output data of the model here, i.e. the price prediction
   vectorf y_norm(1);
//--- run the model
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
     }
//--- do reverse transformation to get the predicted price and to validate it on a ne
   predict=y_norm[0]*s[3]+m[3];
   predictions++;  // increase predictions counter
   Print(predictions,". close prediction = ",predict);
//--- save the bar opening time to check on the next tick
   open_time=time;
  }
```

Compile the Expert Advisor and run testing in the period of year 2022. Specify EURUSD with the H1 timeframe, which is the data on which the model was trained. The tick modeling mode can be ignores, since the code checks the emergence of a new bar.

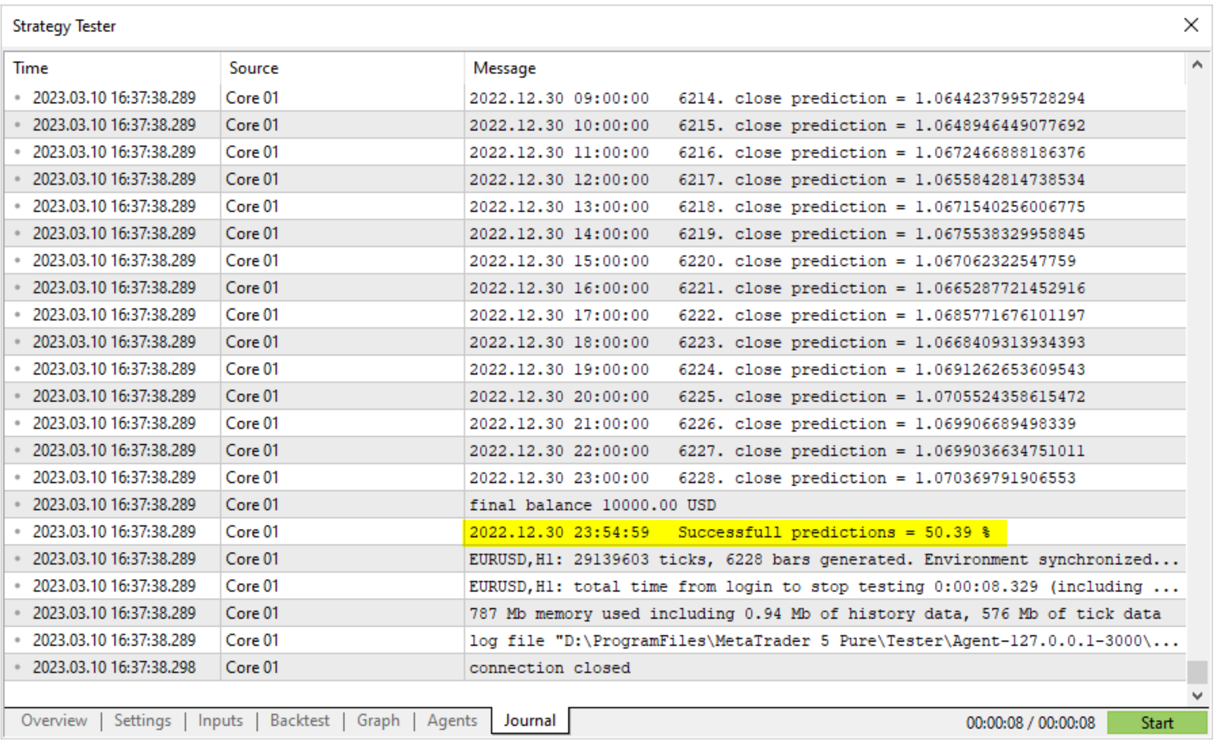


Strategy Tester

Expert: ONNX\PricePrediction_EA.ex5
Symbol: EURUSD H1
Date: Custom period 2022.01.01 2023.01.01
Forward: No 2023.01.25
Delays: Zero latency, ideal execution
Modelling: Every tick
Deposit: 10000 USD 1:100 leverage
Optimization: Disabled

IDE
select a delay to emulate slippage and requotes during trade execution
☐ profit in pips for faster calculations
☐ visual mode with the display of charts, indicators and trades
Overview | Settings | Inputs | Backtest | Graph | Agents | Journal
00:00:08 / 00:00:08
Start

Run and check the result in the testing journal. It shows that a little more than 50% of ppredictions were correct in 2022.



Strategy Tester

```
Time                     Source   Message
2023.03.10 16:37:38.289  Core 01  2022.12.30 09:00:00   6214. close prediction = 1.0644237995728294
2023.03.10 16:37:38.289  Core 01  2022.12.30 10:00:00   6215. close prediction = 1.0648946449077692
2023.03.10 16:37:38.289  Core 01  2022.12.30 11:00:00   6216. close prediction = 1.0672466888186376
2023.03.10 16:37:38.289  Core 01  2022.12.30 12:00:00   6217. close prediction = 1.0655842814738534
2023.03.10 16:37:38.289  Core 01  2022.12.30 13:00:00   6218. close prediction = 1.0671540256006775
2023.03.10 16:37:38.289  Core 01  2022.12.30 14:00:00   6219. close prediction = 1.0675538329958845
2023.03.10 16:37:38.289  Core 01  2022.12.30 15:00:00   6220. close prediction = 1.067062322547759
2023.03.10 16:37:38.289  Core 01  2022.12.30 16:00:00   6221. close prediction = 1.0665287721452916
2023.03.10 16:37:38.289  Core 01  2022.12.30 17:00:00   6222. close prediction = 1.0685771676101197
2023.03.10 16:37:38.289  Core 01  2022.12.30 18:00:00   6223. close prediction = 1.0668409313934393
2023.03.10 16:37:38.289  Core 01  2022.12.30 19:00:00   6224. close prediction = 1.0691262653609543
2023.03.10 16:37:38.289  Core 01  2022.12.30 20:00:00   6225. close prediction = 1.0705524358615472
2023.03.10 16:37:38.289  Core 01  2022.12.30 21:00:00   6226. close prediction = 1.069906689498339
2023.03.10 16:37:38.289  Core 01  2022.12.30 22:00:00   6227. close prediction = 1.0699036634751011
2023.03.10 16:37:38.289  Core 01  2022.12.30 23:00:00   6228. close prediction = 1.070369791906553
2023.03.10 16:37:38.289  Core 01  final balance 10000.00 USD
2023.03.10 16:37:38.289  Core 01  2022.12.30 23:54:59   Successfull predictions = 50.39 %
2023.03.10 16:37:38.289  Core 01  EURUSD,H1: 29139603 ticks, 6228 bars generated. Environment synchronized...
2023.03.10 16:37:38.289  Core 01  EURUSD,H1: total time from login to stop testing 0:00:08.329 (including ...
2023.03.10 16:37:38.289  Core 01  787 Mb memory used including 0.94 Mb of history data, 576 Mb of tick data
2023.03.10 16:37:38.289  Core 01  log file "D:\ProgramFiles\MetaTrader 5 Pure\Tester\Agent-127.0.0.1-3000\...
2023.03.10 16:37:38.298  Core 01  connection closed
```

Overview | Settings | Inputs | Backtest | Graph | Agents | Journal
00:00:08 / 00:00:08
Start

If the preliminary model testing has generated satisfactory results, you can start writing a full-fledged trading strategy based on this model.

OnnxCreate

Create an ONNX session, loading a model from an *.onnx file.

```
long  OnnxCreate(
   string  filename,   // file path
   uint    flags       // flags to create the model
   );
```

Parameters

filename

[in] Path to the *.onnx file of the model relative to the \MQL5\Files\ folder.

flags

[in] Flags from ENUM_ONNX_FLAGS, describing the model creation mode: ONNX_COMMON_FOLDER and ONNX_DEBUG_LOGS.

Return Value

The handle of the created session or INVALID_HANDLE if error occurs. To obtain the error code, call the GetLastError function.

Note

If the specified file is not found on disk, the system retries to open the file, appending the '.onnx' extension to the name.

OnnxCreateFromBuffer

Create an ONNX session, loading a model from a data array.

```
long  OnnxCreateFromBuffer(
   const uchar&  buffer[],    // array reference
   ulong         flags        // model creation flags
   );
```

Parameters

buffer

[in] Array with ONNX model data.

flags

[in] Flags from ENUM_ONNX_FLAGS, describing the model creation mode: ONNX_COMMON_FOLDER and ONNX_DEBUG_LOGS.

Return Value

The handle of the created session or INVALID_HANDLE if error occurs. To obtain the error code, call the GetLastError function.

OnnxRelease

Close an ONNX session.

```
bool  OnnxRelease(
   long    onnx_handle   // ONNX session handle
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

Return Value

Returns true on success; otherwise returns false. To get the error code, call the GetLastError function.

OnnxRun

Run an ONNX model.

```
bool  OnnxRun(
   long     onnx_handle,  // ONNX session handle
   ulong    flags,        // flags describing the run mode
   ...                    // model's inputs and outputs
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

flags

[in] Flags from ENUM_ONNX_FLAGS describing the run mode: ONNX_DEBUG_LOGS and ONNX_NO_CONVERSION.

...

[in] [out] Model inputs and outputs.

Returns true on success or false otherwise. To obtain the error code, call the GetLastError function.

ENUM_ONNX_FLAGS

ID	Description
ONNX_DEBUG_LOGS	Output debug logs
ONNX_NO_CONVERSION	Disable auto conversion, use user data as is
ONNX_COMMON_FOLDER	Load a model file from the Common\Files folder; the value is equal to the FILE_COMMON flag

Example:

```
const long                            ExtOutputShape[] = {1,1};    // model output s
const long                            ExtInputShape [] = {1,10,4}; // model input fo
#resource "Python/model.onnx" as uchar ExtModel[]                  // model as resou
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   matrix rates;
//--- get 10 bars
   if(!rates.CopyRates("EURUSD",PERIOD_H1,COPY_RATES_OHLC,2,10))
      return(-1);
//--- input a set of OHLC vectors
```

```
   matrix x_norm=rates.Transpose();
   vector m=x_norm.Mean(0);
   vector s=x_norm.Std(0);
   matrix mm(10,4);
   matrix ms(10,4);
//--- fill in the normalization matrices
   for(int i=0; i<10; i++)
     {
      mm.Row(m,i);
      ms.Row(s,i);
     }
//--- normalize the input data
   x_norm-=mm;
   x_norm/=ms;
//--- create the model
   long handle=OnnxCreateFromBuffer(ExtModel,ONNX_DEBUG_LOGS);
//--- specify the shape of the input data
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("OnnxSetInputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- specify the shape of the output data
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("OnnxSetOutputShape failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- convert normalized input data to float type
   matrixf x_normf;
   x_normf.Assign(x_norm);
//--- get the output data of the model here, i.e. the price prediction
   vectorf y_norm(1);
//--- run the model
   if(!OnnxRun(handle,ONNX_DEBUG_LOGS | ONNX_NO_CONVERSION,x_normf,y_norm))
     {
      Print("OnnxRun failed, error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- print the output value of the model to the log
   Print(y_norm);
//--- do the reverse transformation to get the predicted price
   double y_pred=y_norm[0]*s[3]+m[3];
   Print("price predicted:",y_pred);
//--- completed operation
   OnnxRelease(handle);
```

```
    return(0);
   };
```

See also

OnnxSetInputShape, OnnxSetOutputShape

OnnxGetInputCount

Get the number of inputs in an ONNX model.

```
long  OnnxGetInputCount(
   long    onnx_handle   // ONNX session handle
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

Return Value

Returns the number of input parameters on success; otherwise returns -1. To get the error code, call the GetLastError function.

OnnxGetOutputCount

Get the number of outputs in an ONNX model.

```
long  OnnxGetOutputCount(
   long   onnx_handle   // ONNX session handle
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

Return Value

Returns the number of output parameters on success; otherwise returns -1. To get the error code, call the GetLastError function.

OnnxGetInputName

Get the name of a model's input by index.

```
string  OnnxGetInputName(
   long   onnx_handle,   // ONNX session handle
   long   index          // parameter index
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

index

[in] Index of the input parameter, starting with 0.

Return Value

Returns the name of the input parameter on success; otherwise returns NULL. To get the error code, call the GetLastError function.

OnnxGetOutputName

Get the name of a model's output by index.

```
string  OnnxGetOutputName(
   long   onnx_handle,   // ONNX session handle
   long   index          // parameter index
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

index

[in] Index of the output parameter, starting with 0.

Return Value

Returns the name of the output parameter on success; otherwise returns NULL. To get the error code, call the GetLastError function.

OnnxGetInputTypeInfo

Get the description of the input type from the model.

```
bool  OnnxGetInputTypeInfo(
   long            onnx_handle,   // ONNX session handle
   long            index,         // parameter index
   OnnxTypeInfo&   typeinfo       // parameter type description
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

index

[in] Index of the input parameter, starting with 0.

typeinfo

[out] The OnnxTypeInfo structure that describes the type of the input parameter.

Return Value

Returns true on success; otherwise returns false. To get the error code, call the GetLastError function.

OnnxGetOutputTypeInfo

Get the description of the output type from the model.

```
bool  OnnxGetOutputTypeInfo(
   long            onnx_handle,   // ONNX session handle
   long            index,         // parameter index
   OnnxTypeInfo&   typeinfo       // parameter type description
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

index

[in] Index of the output parameter, starting with 0.

typeinfo

[out] The OnnxTypeInfo structure that describes the type of the output parameter.

Return Value

Returns true on success; otherwise returns false. To get the error code, call the GetLastError function.

OnnxSetInputShape

Set the shape of a model's input data by index.

```
bool  OnnxSetInputShape(
   long           onnx_handle,   // ONNX session handle
   long           input_index,   // input parameter index
   const ulong&   shape[]        // array describing input data shape
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

input_index

[in] Index of the input parameter, starting from 0.

shape

[in] Array describing model's input data shape.

Return Value

Returns the name of the input parameter on success; otherwise returns NULL. To get the error code, call the GetLastError function.

Example:

```
//---- describe the shapes of the model's input and output data
   const long  ExtOutputShape[] = {1,1};
   const long  ExtInputShape [] = {1,10,4};
//--- create the model
   long handle=OnnxCreateFromBuffer(model,ONNX_DEBUG_LOGS);
//--- specify the shape of the input data
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("failed, OnnxSetInputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- specify the shape of the output data
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("failed, OnnxSetOutputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
```

See also

OnnxSetOutputShape

OnnxSetOutputShape

Set the shape of a model's output data by index.

```
bool  OnnxSetOutputShape(
   long           onnx_handle,    // ONNX session handle
   long           output_index,   // output parameter index
   const ulong&   shape[]         // array describing output data shape
   );
```

Parameters

onnx_handle

[in] ONNX session object handle created via OnnxCreate or OnnxCreateFromBuffer.

output_index

[in] Index of the output parameter, starting from 0.

shape

[in] Array describing model's output data shape.

Return Value

Returns the name of the input parameter on success; otherwise returns NULL. To get the error code, call the GetLastError function.

Example:

```
//---- describe the shapes of the model's input and output data
   const long  ExtOutputShape[] = {1,1};
   const long  ExtInputShape [] = {1,10,4};
//--- create the model
   long handle=OnnxCreateFromBuffer(model,ONNX_DEBUG_LOGS);
//--- specify the shape of the input data
   if(!OnnxSetInputShape(handle,0,ExtInputShape))
     {
      Print("failed, OnnxSetInputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
//--- specify the shape of the output data
   if(!OnnxSetOutputShape(handle,0,ExtOutputShape))
     {
      Print("failed, OnnxSetOutputShape error ",GetLastError());
      OnnxRelease(handle);
      return(-1);
     }
```

See also

OnnxSetInputShape

Data structures

The following data structures are used for operations with ONNX models:

OnnxTypeInfo

The structure describes the type of an input or output parameter of an ONNX model

```
struct OnnxTypeInfo
  {
   ENUM_ONNX_TYPE         type;           // parameter type
   OnnxTensorTypeInfo     tensor;         // tensor description
   OnnxMapTypeInfo        map;            // map description
   OnnxSequenceTypeInfo   sequence;       // sequence description
  };
```

Only tensor (ONNX_TYPE_TENSOR) can be used as an input. In this case, only the OnnxTypeInfo::tensor field is filled with values, while other fields (map and sequence) are not defined.

Only one of the three OnnxTypeInfo types (ONNX_TYPE_TENSOR, ONNX_TYPE_MAP or ONNX_TYPE_SEQUENCE) can be used as an input. The corresponding substructure (OnnxTypeInfo::tensor, OnnxTypeInfo::map or OnnxTypeInfo::sequence) is filled depending on the type.

OnnxTensorTypeInfo

The structure describes the tensor in the input or output parameter of an ONNX model

```
struct OnnxTensorTypeInfo
  {
   const ENUM_ONNX_DATA_TYPE    data_type;       // data type in the tensor
   const long                   dimensions[];    // number of elements in the tensor
  };
```

OnnxMapTypeInfo

The structure describes the map obtained in the output parameter of an ONNX model

```
struct OnnxMapTypeInfo
  {
   const ENUM_ONNX_DATA_TYPE    key_type;        // key type
   const OnnxTypeInfo &         value_type;      // value type
  };
```

OnnxSequenceTypeInfo

The structure describes the sequence obtained in the output parameter of an ONNX model

```
struct OnnxSequenceTypeInfo
```

```
{
 const OnnxTypeInfo&         value_type;       // data type in the sequence
};
```

ENUM_ONNX_TYPE

The **ENUM_ONNX_TYPE** enumeration describes the type of a model parameter

ID	Description
ONNX_TYPE_UNKNOWN	Unknown
ONNX_TYPE_TENSOR	Tensor
ONNX_TYPE_SEQUENCE	Sequence
ONNX_TYPE_MAP	Map
ONNX_TYPE_OPAQUE	Abstract (opaque)
ONNX_TYPE_SPARSETENSOR	Sparse tensor

ENUM_ONNX_DATA_TYPE

The **ENUM_ONNX_DATA_TYPE** enumeration describes the type of data used

ID	Description
ONNX_DATA_TYPE_UNDEFINED	Undefined
ONNX_DATA_TYPE_FLOAT	float
ONNX_DATA_TYPE_INT8	8-bit int
ONNX_DATA_TYPE_UINT16	16-bit uint
ONNX_DATA_TYPE_INT16	16-bit int
ONNX_DATA_TYPE_INT32	32-bit int
ONNX_DATA_TYPE_INT64	64-bit int
ONNX_DATA_TYPE_STRING	string
ONNX_DATA_TYPE_BOOL	bool
ONNX_DATA_TYPE_FLOAT16	16-bit float
ONNX_DATA_TYPE_DOUBLE	double
ONNX_DATA_TYPE_UINT32	32-bit uint
ONNX_DATA_TYPE_UINT64	64-bit uint

ID	Description
ONNX_DATA_TYPE_COMPLEX64	64-bit complex number
ONNX_DATA_TYPE_COMPLEX128	128-bit complex number
ONNX_DATA_TYPE_BFLOAT16	16-bit bfloat (Brain Floating Point)

ENUM_ONNX_FLAGS

The **ENUM_ONNX_FLAGS** enumeration describes the model run mode

ID	Description
ONNX_DEBUG_LOGS	Output debug logs
ONNX_NO_CONVERSION	Disable auto conversion, use user data as is
ONNX_COMMON_FOLDER	Load a model file from the Common\Files folder; the value is equal to the FILE_COMMON flag

Array conversion when working with ONNX models

Machine learning tasks do not always require greater computational accuracy. To speed up calculations, some models use lower-precision data types such as Float16 and even Float8. To allow users to input the relevant data into models, MQL5 provides four special functions which convert standard MQL5 types into special FP16 and FP8 types.

Function	Action
ArrayToFP16	Copies an array of type float or double into an array of type ushort with the given format
ArrayToFP8	Copies an array of type float or double into an array of type uchar with the given format
ArrayFromFP16	Copies an array of type ushort into an array of float or double type with the given format
ArrayFromFP8	Copies an array of type uchar into an array of float or double type with the given format

These array conversion functions use special formats specified in the below enumerations.

ENUM_FLOAT16_FORMAT

The **ENUM_FLOAT16_FORMAT** enumeration describes two FP16 type formats.

ID	Description
FLOAT_FP16	Standard 16-bit format, also known as half
FLOAT_BFP16	Special brain float point format

Each of these formats has its advantages and limitations. FLOAT16 provides higher accuracy but requires more storage and computation resources. BFLOAT16, on the other hand, provides higher performance and efficiency in data processing, but may be less accurate.

ENUM_FLOAT8_FORMAT

The **ENUM_FLOAT8_FORMAT** enumeration describes four FP8 type formats.

FP8 (8-bit floating point) is one of the data types used to represent floating point numbers. In FP8, each number is represented by 8 data bits, typically divided into three components: sign, exponent and mantissa. This format offers a balance between accuracy and storage efficiency, making it attractive for applications that require memory and computational efficiency.

ID	Description
FLOAT_FP8_E4M3FN	8-bit floating point number, 4 bits for the exponent and 3 bits for the mantissa. Typically used as coefficients.
FLOAT_FP8_E4M3FNUZ	8-bit floating point number, 4 bits for the exponent and 3 bits for the mantissa. Supports NaN, does not support negative zero and Inf. Typically used as coefficients.
FLOAT_FP8_E5M2FN	8-bit floating point number, 5 bits for the exponent and 2 bits for the mantissa. Supports NaN and Inf. Typically used for gradients.
FLOAT_FP8_E5M2FNUZ	8-bit floating point number, 5 bits for the exponent and 2 bits for the mantissa. Supports NaN, does not support negative zero and Inf. Also used for gradients.

One of the key advantages of FP8 is its efficiency in processing large datasets. By employing compact number representation, FP8 reduces memory requirements and accelerates calculations. This is especially important in machine learning and artificial intelligence applications which often process large datasets.

Standard Library

This group of chapters contains the technical details of the MQL5 Standard Library and descriptions of all its key components.

MQL5 Standard Library is written in MQL5 and is designed to facilitate writing programs (indicators, scripts, experts) for end users. Library provides convenient access to the most of the internal MQL5 functions.

MQL5 Standard Library is placed in the working directory of the terminal in the 'Include' folder.

Section	Location
Mathematics	Include\Math\
OpenCL	Include\OpenCL\
Basic Class CObject	Include\
Data Collections	Include\Arrays\
Generic Data Collections	Include\Generic\
Files	Include\Files\
Strings	Include\Strings\
Graphic Objects	Include\Objects\
Custom Graphics	Include\Canvas\
3D Graphics	Include\Canvas\
Price Charts	Include\Charts\
Scientific Charts	Include\Graphics\
Indicators	Include\Indicators\
Trade classes	Include\Trade\
Strategy Modules	Include\Expert\
Panels and Dialogs	Include\Controls\

Mathematics

Multiple libraries are provided to perform calculations in various areas of mathematics:

- Statistics – functions for working with various distributions of the Probability theory
- Fuzzy logic – library that implements Mamdani and Sugeno fuzzy inference systems
- ALGLIB – data analysis (clustering, decision trees, linear regression, neural networks), solving differential equations, Fourier transform, numerical integration, optimization problems, statistical analysis, and more.

Statistics

The Statistical Library provides a convenient way of working with the basic statistical distributions.

The library provides 5 functions for each distribution:

1. Calculation of probability density – functions of the form MathProbabilityDensityX()
2. Calculation of probabilities – functions of the form MathCumulativeDistributionX()
3. Calculation of distribution quantiles – functions of the form MathQuantileX()
4. Generation of random numbers with the specified distribution – functions of the form MathRandomX()
5. Calculation of the theoretical moments of the distributions – functions of the form MathMomentsX()

In addition to calculation of values for the individual random variables, the library also provides overloads for the functions, which perform the same calculations for arrays.

- Statistical Characteristics
- Normal Distribution
- Log-normal distribution
- Beta distribution
- Noncentral beta distribution
- Gamma distribution
- Chi-squared distribution
- Noncentral chi-squared distribution
- Exponential distribution
- F-distribution
- Noncentral F-distribution
- t-distribution
- Noncentral t-distribution
- Logistic distribution
- Cauchy distribution
- Uniform distribution
- Weibull distribution
- Binomial distribution
- Negative binomial distribution
- Geometric distribution
- Hypergeometric distribution
- Poisson distribution
- Subfunctions

Example:

```
//+------------------------------------------------------------------+
//|                                    NormalDistributionExample.mq5 |
//|                         Copyright 2016, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- include the functions for calculating the normal distribution
#include <Math\Stat\Normal.mqh>
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the parameters of the normal distribution
   double mu=5.0;
   double sigma=1.0;
   PrintFormat("Normal distribution with parameters mu=%G and sigma=%G, calculation ex
//--- set the interval
   double x1=mu-sigma;
   double x2=mu+sigma;
//--- variables for probability calculation
   double cdf1,cdf2,probability;
//--- variables for error codes
   int error_code1,error_code2;
//--- calculate the values of distribution functions
   cdf1=MathCumulativeDistributionNormal(x1,mu,sigma,error_code1);
   cdf2=MathCumulativeDistributionNormal(x2,mu,sigma,error_code2);
//--- check the error codes
   if(error_code1==ERR_OK && error_code2==ERR_OK)
     {
      //--- calculate probability of a random variable in the range
      probability=cdf2-cdf1;
      //--- output the result
      PrintFormat("1. Calculate probability of a random variable within the range of %
      PrintFormat("  Answer: Probability = %5.8f",probability);
     }

//--- Find the value range of random variable x, corresponding to the 95% confidence
   probability=0.95; //  set the confidence probability
//--- set the probabilities at the interval bounds
   double p1=(1.0-probability)*0.5;
   double p2=probability+(1.0-probability)*0.5;
//--- calculate the interval bounds
   x1=MathQuantileNormal(p1,mu,sigma,error_code1);
   x2=MathQuantileNormal(p2,mu,sigma,error_code2);
//--- check the error codes
   if(error_code1==ERR_OK && error_code2==ERR_OK)
     {
      //--- output the result
      PrintFormat("2. For confidence interval = %.2f, find the range of random variabl
      PrintFormat("  Answer: range is  %5.8f <= x <=%5.8f",x1,x2);
     }

   PrintFormat("3. Compute the first 4 calculated and theoretical moments of the distr
//--- Generate an array of random numbers, calculate the first 4 moments and compare
   int data_count=1000000;  // set the number of values and prepare an array
   double data[];
   ArrayResize(data,data_count);
//--- generate random values and store them into the array
```

```
   for(int i=0; i<data_count; i++)
     {
      data[i]=MathRandomNormal(mu,sigma,error_code1);
     }
//--- set the index of the initial value and the amount of data for calculation
   int start=0;
   int count=data_count;
//--- calculate the first 4 moments of the generated values
   double mean=MathMean(data,start,count);
   double variance=MathVariance(data,start,count);
   double skewness=MathSkewness(data,start,count);
   double kurtosis=MathKurtosis(data,start,count);
//--- variables for the theoretical moments
   double normal_mean=0;
   double normal_variance=0;
   double normal_skewness=0;
   double normal_kurtosis=0;
//--- display the values of the calculated moments
   PrintFormat("            Mean           Variance          Skewness           Kurtos
   PrintFormat("Calculated  %.10f   %.10f      %.10f      %.10f",mean,variance,skewnes
//--- calculate the theoretical values of the moments and compare them with the obtai
   if(MathMomentsNormal(mu,sigma,normal_mean,normal_variance,normal_skewness,normal_ku
     {
      PrintFormat("Theoretical %.10f   %.10f      %.10f       %.10f",normal_mean,norma
      PrintFormat("Difference  %.10f   %.10f      %.10f       %.10f",mean-normal_mean,
     }
  }
```

Statistical Characteristics

This group of functions calculates the Statistical Characteristics of the array elements:

- mean,
- variance,
- skewness,
- kurtosis,
- median,
- root-mean-square and
- standard deviation.

Function	Description
MathMean	Calculates the mean (first moment) of array elements
MathVariance	Calculates the variance (second moment) of array elements
MathSkewness	Calculates the skewness (third moment) of array elements
MathKurtosis	Calculates the kurtosis (fourth moment) of array elements
MathMoments	Calculates the first 4 moments (mean, variance, skewness, kurtosis) of array elements
MathMedian	Calculates the median value of array elements
MathStandardDeviation	Calculates the standard deviation of array elements
MathAverageDeviation	Calculates the average absolute deviation of array elements

MathMean

Calculates the mean (first moment) of array elements. Analog of the mean() in R.

```
double  MathMean(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation of the mean.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The mean of array elements. In case of error it returns NaN (not a number).

MathVariance

Calculates the variance (second moment) of array elements. Analog of the var() in R.

```
double  MathVariance(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Variance of array elements. In case of error it returns NaN (not a number).

MathSkewness

Calculates the skewness (third moment) of array elements. Analog of the skewness() in R (e1071 library).

```
double  MathSkewness(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Skewness of array elements. In case of error it returns NaN (not a number).

MathKurtosis

Calculates the kurtosis (fourth moment) of array elements. Analog of the kurtosis() in R (e1071 library).

```
double  MathKurtosis(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Kurtosis of array elements. In case of error it returns NaN (not a number).

Disclaimer

Calculation of the kurtosis is performed using the excess kurtosis around the normal distribution (excess kurtosis=kurtosis-3), i.e. the excess kurtosis of a normal distribution is zero.

It is positive if the peak of the distribution around the expected value is sharp, and negative if the peak is flat.

MathMoments

Calculates the first 4 moments (mean, variance, skewness, kurtosis) of array elements.

```
double  MathMoments(
   const double&  array[],              // array with data
   double&        mean,                 // mean (1st moment)
   double&        variance,             // variance (2nd moment)
   double&        skewness,             // skewness (3rd moment)
   double&        kurtosis,             // kurtosis (4th moment)
   const int      start=0,              // initial index
   const int      count=WHOLE_ARRAY     // the number of elements
   );
```

Parameters

array

[in] Array with data for calculation.

mean

[out] Variable for the mean (1st moment)

variance

[out] Variable for the variance (2nd moment)

skewness

[out] Variable for the skewness (3rd moment)

kurtosis

[out] Variable for the kurtosis (4th moment)

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

Disclaimer

Calculation of the kurtosis is performed using the excess kurtosis around the normal distribution (excess kurtosis=kurtosis-3), i.e. the excess kurtosis of a normal distribution is zero.

It is positive if the peak of the distribution around the expected value is sharp, and negative if the peak is flat.

MathMedian

Calculates the median value of array elements. Analog of the median() in R.

```
double  MathMedian(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The median value of array elements. In case of error it returns NaN (not a number).

MathStandardDeviation

Calculates the standard deviation of array elements. Analog of the sd() in R.

```
double  MathStandardDeviation(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The standard deviation of array elements. In case of error it returns NaN (not a number).

MathAverageDeviation

Calculates the average absolute deviation of array elements. Analog of the aad() in R.

```
double  MathAverageDeviation(
   const double&   array[]                // array with data
   );
```

Parameters

array

[in] Array with data for calculation.

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

The average absolute deviation of array elements. In case of error it returns NaN (not a number).

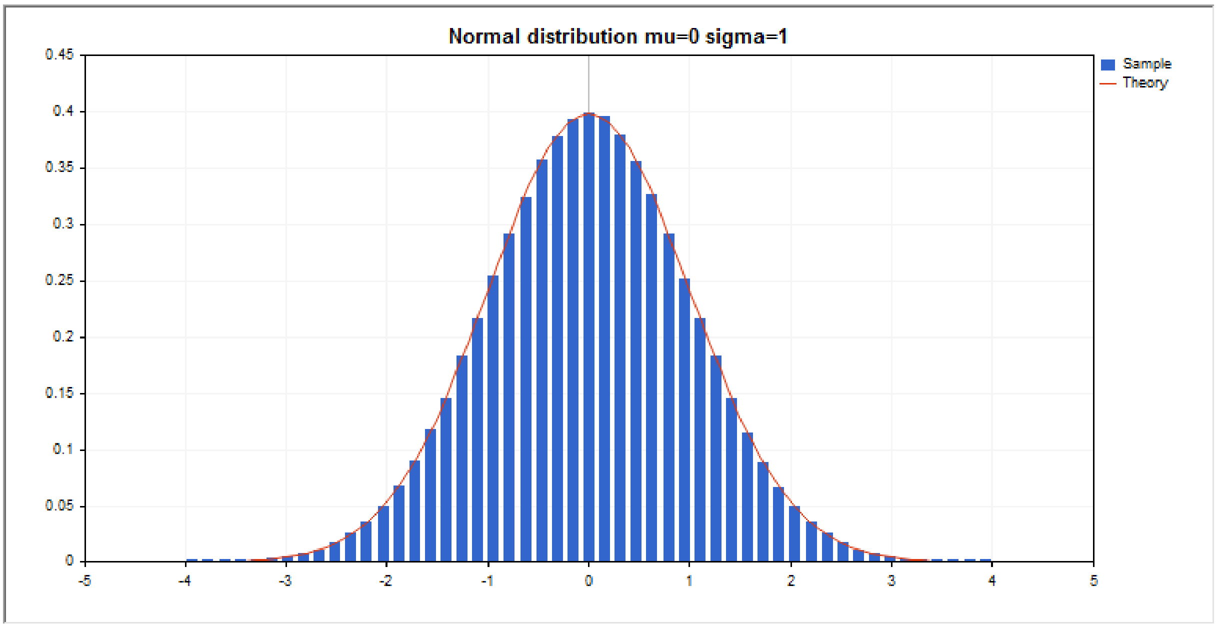
Normal Distribution

This section contains functions for working with normal distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the normal law. The distribution is defined by the following formula:

$$f_{Normal}(x|\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- x — value of the random variable
- μ — expected value
- σ — root-mean-square deviation



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNormal	Calculates the probability density function of the normal distribution
MathCumulativeDistributionNormal	Calculates the value of the normal probability distribution function
MathQuantileNormal	Calculates the value of the inverse normal distribution function for the specified probability
MathRandomNormal	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the normal law

Function	Description
MathMomentsNormal	Calculates the theoretical numerical values of the first 4 moments of the normal distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Normal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=0;  // expected value  (mean)
input double std_dev=1;     // root-mean-square deviation (standard deviation)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the normal distribution
   MathRandomNormal(mean_value,std_dev,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNormal(x2,mean_value,std_dev,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Normal distribution mu=%G sigma=%G",mean_value
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
//--- plot all curves
   graphic.CurvePlotAll();
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

```
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityNormal

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNormal(
   const double  x,                // value of random variable
   const double  mu,               // mean parameter of the distribution (expected value
   const double  sigma,            // sigma parameter of the distribution (root-mean-squ
   const bool    log_mode,         // calculate the logarithm of the value
   int&          error_code        // variable to store the error code
   );
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNormal(
   const double  x,                // value of random variable
   const double  mu,               // mean parameter of the distribution (expected value
   const double  sigma,            // sigma parameter of the distribution (root-mean-squ
   int&          error_code        // variable to store the error code
   );
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the dnorm() in R.

```
bool  MathProbabilityDensityNormal(
   const double&  x[],              // array with the values of random variable
   const double   mu,               // mean parameter of the distribution (expected valu
   const double   sigma,            // sigma parameter of the distribution (root-mean-sq
   const bool     log_mode,         // calculate the logarithm of the value
   double&        result[]          // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNormal(
   const double&  x[],              // array with the values of random variable
   const double   mu,               // mean parameter of the distribution (expected valu
   const double   sigma,            // sigma parameter of the distribution (root-mean-sq
   double&        result[]          // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of the probability density function.

MathCumulativeDistributionNormal

Calculates the value of the normal distribution function with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNormal(
   const double  x,               // value of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   const bool    tail,            // flag for calculation of tail
   const bool    log_mode,        // calculate the logarithm of the value
   int&          error_code       // variable to store the error code
   );
```

Calculates the value of the normal distribution function with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNormal(
   const double  x,               // value of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   int&          error_code       // variable to store the error code
   );
```

Calculates the value of the normal distribution function with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the dnorm() in R.

```
bool  MathCumulativeDistributionNormal(
   const double& x[],             // array with the values of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   const bool    tail,            // flag for calculation of tail
   const bool    log_mode,        // calculate the logarithm of the value
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the normal distribution function with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNormal(
   const double& x[],             // array with the values of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

tail

[in] Flag of calculation. If tail=true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of the probability function.

MathQuantileNormal

For the specified *probability*, the function calculates the value of inverse normal distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileNormal(
   const double  probability,     // probability value of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   const bool    tail,            // flag for calculation of tail
   const bool    log_mode,        // calculate the logarithm of the value
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse normal distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileNormal(
   const double  probability,     // probability value of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse normal distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the qnorm() in R.

```
bool  MathQuantileNormal(
   const double& probability[],   // array with probability values of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   const bool    tail,            // flag for calculation of tail
   const bool    log_mode,        // calculate the logarithm of the value
   double&       result[]         // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse normal distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool  MathQuantileNormal(
   const double& probability[],   // array with probability values of random variable
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   double&       result[]         // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

tail

[in] Flag of calculation. If false, then calculation is performed for 1.0 - probability.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the quantiles.

MathRandomNormal

Generates a pseudorandom variable distributed according to the normal law with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathRandomNormal(
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   int&          error_code       // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the normal law with the mu and sigma parameters. In case of error it returns false. Analog of the rnorm() in R.

```
bool  MathRandomNormal(
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   const int     data_count,      // amount of required data
   double&       result[]         // array to obtain the pseudorandom variables
   );
```

Parameters

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

data_count

[in] The number of pseudorandom variables to be obtained.

error_code

[out] Variable to get the error code.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNormal

Calculates the theoretical numerical values of the first 4 moments of the normal distribution.

```
double  MathMomentsNormal(
   const double  mu,              // expected value
   const double  sigma,           // root-mean-square deviation
   double&       mean,            // variable for the mean
   double&       variance,        // variable for the variance
   double&       skewness,        // variable for the skewness
   double&       kurtosis,        // variable for the kurtosis
   int&          error_code       // variable to store the error code
   );
```

Parameters

mu

[in] mean parameter of the distribution (expected value).

sigma

[in] sigma parameter of the distribution (root-mean-square deviation).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

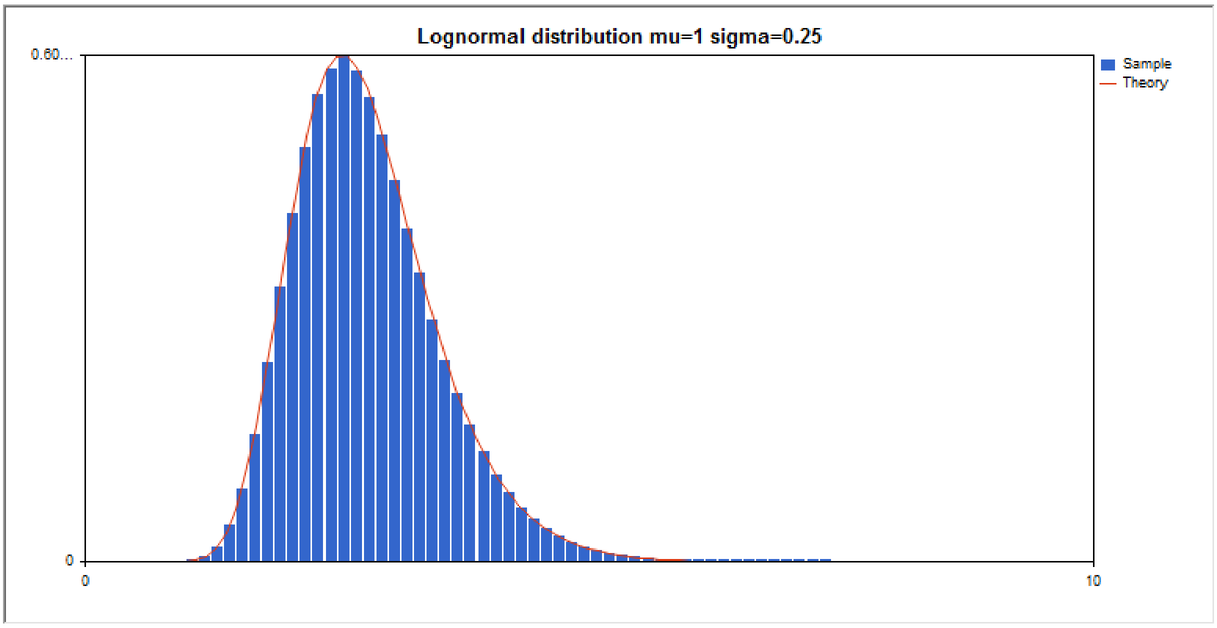
Log-normal distribution

This section contains functions for working with log-normal distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the log-normal law. The log-normal distribution is defined by the following formula:

$$f_{Lognormal}(x \mid \mu,\sigma)=\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$$

where:

- x — value of the random variable
- µ — logarithm of the expected value
- σ — logarithm of the root-mean-square deviation



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityLognormal	Calculates the probability density function of the log-normal distribution
MathCumulativeDistributionLognormal	Calculates the value of the log-normal probability distribution function
MathQuantileLognormal	Calculates the value of the inverse log-normal distribution function for the specified probability
MathRandomLognormal	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the log-normal law
MathMomentsLognormal	Calculates the theoretical numerical values of the first 4 moments of the log-normal distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Lognormal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=1.0;  // logarithm of the expected value (log mean)
input double std_dev=0.25;    // logarithm of the root-mean-square deviation (log star
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the log-normal distribution
   MathRandomLognormal(mean_value,std_dev,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityLognormal(x2,mean_value,std_dev,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Lognormal distribution mu=%G sigma=%G",mean_va
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(theor_max);
   graphic.YAxis().Min(0);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                          |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityLognormal

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityLognormal(
   const double  x,                // value of random variable
   const double  mu,               // logarithm of the expected value (log mean)
   const double  sigma,            // logarithm of the root-mean-square deviation (log
   const bool    log_mode,         // calculate the logarithm of the value, if log_mode
   int&          error_code        // variable to store the error code
   );
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityLognormal(
   const double  x,                // value of random variable
   const double  mu,               // logarithm of the expected value (log mean)
   const double  sigma,            // logarithm of the root-mean-square deviation (log
   int&          error_code        // variable to store the error code
   );
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns NaN. Analog of the dlnorm() in R.

```
bool  MathProbabilityDensityLognormal(
   const double&  x[],             // array with the values of random variable
   const double   mu,              // logarithm of the expected value (log mean)
   const double   sigma,           // logarithm of the root-mean-square deviation (log
   const bool     log_mode,        // calculate the logarithm of the value, if log_mode
   double&        result[]         // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of log-normal distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityLognormal(
   const double&  x[],             // array with the values of random variable
   const double   mu,              // logarithm of the expected value (log mean)
   const double   sigma,           // logarithm of the root-mean-square deviation (log
   double&        result[]         // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of the probability density function.

MathCumulativeDistributionLognormal

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionLognormal(
   const double  x,                // value of random variable
   const double  mu,               // logarithm of the expected value (log mean)
   const double  sigma,            // logarithm of the root-mean-square deviation (log s
   const bool    tail,             // flag of calculation, if true, then the probability
   const bool    log_mode,         // calculate the logarithm of the value, if log_mode=
   int&          error_code        // variable to store the error code
   );
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionLognormal(
   const double  x,                // value of random variable
   const double  mu,               // logarithm of the expected value (log mean)
   const double  sigma,            // logarithm of the root-mean-square deviation (log s
   int&          error_code        // variable to store the error code
   );
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the plnorm() in R.

```
bool  MathCumulativeDistributionLognormal(
   const double&  x[],              // array with the values of random variable
   const double   mu,               // logarithm of the expected value (log mean)
   const double   sigma,            // logarithm of the root-mean-square deviation (log
   const bool     tail,             // flag of calculation, if true, then the probabilit
   const bool     log_mode,         // flag to calculate the logarithm of the value, if
   double&        result[]          // array for values of the probability function
   );
```

Calculates the log-normal distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionLognormal(
   const double&  x[],              // array with the values of random variable
   const double   mu,               // logarithm of the expected value (log mean)
   const double   sigma,            // logarithm of the root-mean-square deviation (log
   double&        result[]          // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of the probability function.

MathQuantileLognormal

For the specified probability, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileLognormal(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // logarithm of the expected value (log mean)
   const double  sigma,          // logarithm of the root-mean-square deviation (log s
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified probability, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileLognormal(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // logarithm of the expected value (log mean)
   const double  sigma,          // logarithm of the root-mean-square deviation (log s
   int&          error_code      // variable to store the error code
   );
```

For the specified probability[] array of probability values, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the qlnorm() in R.

```
bool  MathQuantileLognormal(
   const double&  probability[],  // array with probability values of random variable
   const double   mu,             // logarithm of the expected value (log mean)
   const double   sigma,          // logarithm of the root-mean-square deviation (log
   const bool     tail,           // flag of calculation, if false, then calculation i
   const bool     log_mode,       // flag of calculation, if log_mode=true, calculatio
   double&        result[]        // array with values of quantiles
   );
```

For the specified probability[] array of probability values, the function calculates the value of inverse log-normal distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool  MathQuantileLognormal(
   const double&  probability[],  // array with probability values of random variable
   const double   mu,             // logarithm of the expected value (log mean)
   const double   sigma,          // logarithm of the root-mean-square deviation (log
   double&        result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable occurrence.

probability[]

[in] Array with probability values of random variable.

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to store the error code.

result[]

[out] Array with values of quantiles.

MathRandomLognormal

Generates a pseudorandom variable distributed according to the log-normal law with the mu sigma parameters. In case of error it returns NaN.

```
double  MathRandomLognormal(
   const double  mu,            // logarithm of the expected value (log mean)
   const double  sigma,         // logarithm of the root-mean-square deviation (log s
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the log-normal law with the mu sigma parameters. In case of error it returns false. Analog of the rlnorm() in R.

```
double  MathRandomLognormal(
   const double  mu,            // logarithm of the expected value (log mean)
   const double  sigma,         // logarithm of the root-mean-square deviation (log s
   const int     data_count,    // amount of required data
   double&       result[]       // array with values of pseudorandom variables
   );
```

Parameters

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

data_count

[in] Amount of required data.

error_code

[out] Variable to store the error code.

result[]

[out] Array with values of pseudorandom variables.

MathMomentsLognormal

Calculates the theoretical numerical values of the first 4 moments of the log-normal distribution. Returns true if calculation of the moments has been successful, otherwise false.

```
double  MathMomentsLognormal(
   const double  mu,              // logarithm of the expected value (log mean)
   const double  sigma,           // logarithm of the root-mean-square deviation (log
   double&       mean,            // variable for the mean
   double&       variance,        // variable for the variance
   double&       skewness,        // variable for the skewness
   double&       kurtosis,        // variable for the kurtosis
   int&          error_code       // variable to store the error code
   );
```

Parameters

mu

[in] Logarithm of the expected value (log_mean).

sigma

[in] Logarithm of the root-mean-square deviation (log standard deviation).

mean

[in] Variable for the mean.

variance

[out] Variable for the variance.

skewness

[out] Variable for the skewness.

kurtosis

[out] Variable for the kurtosis.

error code

[out] Variable to store the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

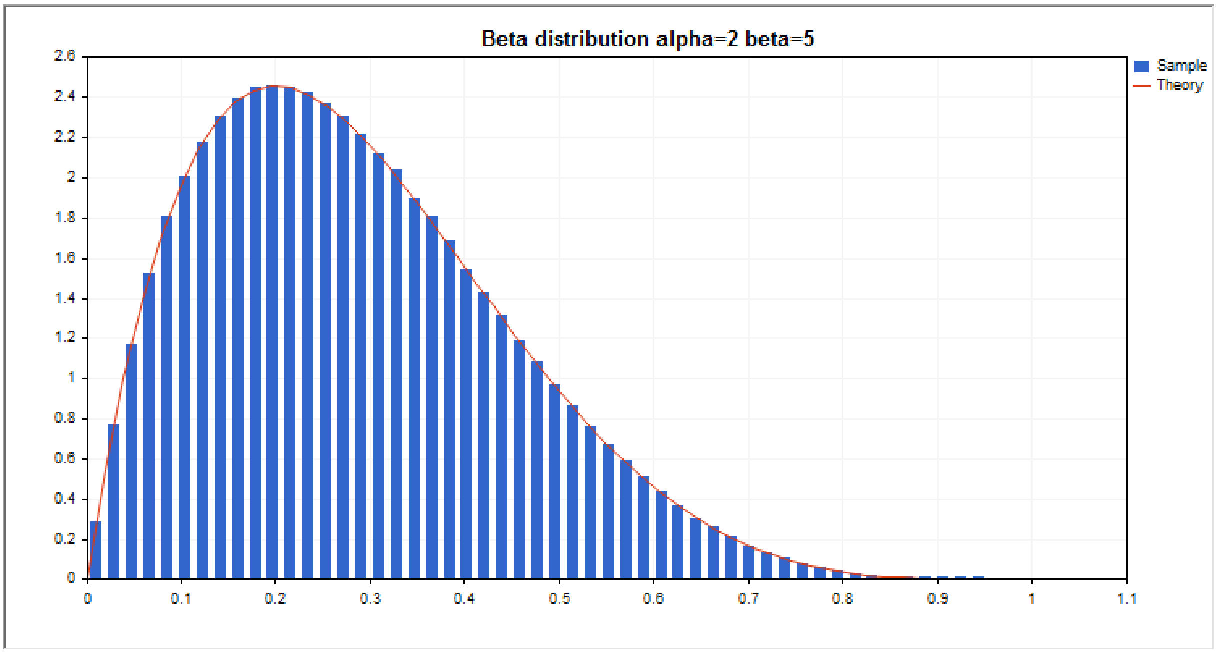
Beta distribution

This section contains functions for working with beta distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The beta distribution is defined by the following formula:

$$f_{Beta}(x \mid a,b) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}$$

where:

- x — value of the random variable
- a — the first parameter of beta distribution
- b — the second parameter of beta distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityBeta	Calculates the probability density function of the beta distribution
MathCumulativeDistributionBeta	Calculates the value of the beta probability distribution function
MathQuantileBeta	Calculates the value of the inverse beta distribution function for the specified probability
MathRandomBeta	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the beta distribution law
MathMomentsBeta	Calculates the theoretical numerical values of the first 4 moments of the beta distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Beta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=2;   // the first parameter of beta distribution (shape1)
input double beta=5;    // the second parameter of beta distribution (shape2)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the beta distribution
   MathRandomBeta(alpha,beta,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityBeta(x2,alpha,beta,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Beta distribution alpha=%G beta=%G",alpha,beta
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
```

```
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityBeta

Calculates the value of the probability density function of beta distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the dbeta() in R.

```
bool  MathProbabilityDensityBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2
   const bool    log_mode,        // flag to calculate the logarithm of the value, if
   double&       result[]         // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2
   double&       result[]         // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionBeta

Calculates the probability distribution function of beta distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of beta distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the pbeta() in R.

```
bool  MathCumulativeDistributionBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2
   const bool    tail,            // flag of calculation, if true, then the probabilit
   const bool    log_mode,        // flag to calculate the logarithm of the value, if 
   double&       result[]         // array for values of the probability function
   );
```

<t0>Calculates the probability distribution function of beta distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileBeta

For the specified *probability*, the function calculates the value of inverse beta distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileBeta(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse beta distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileBeta(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse beta distribution function with the a and b parameters. In case of error it returns false. Analog of the qbeta() in R.

```
double  MathQuantileBeta(
   const double&  probability[], // array with probability values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   double&       result[]         // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the values of inverse beta distribution function with the a and b parameters. In case of error it returns false.

```
bool  MathQuantileBeta(
   const double& probability[],  // array with probability values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   double&       result[]         // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

tail

[in] Flag of calculation, if lower_tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomBeta

Generates a pseudorandom variable distributed according to the law of beta distribution with the a and b parameters. In case of error it returns NaN.

```
double  MathRandomBeta(
   const double  a,             // the first parameter of beta distribution (shape1)
   const double  b,             // the second parameter of beta distribution (shape2)
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of beta distribution with the a and b parameters. In case of error it returns false. Analog of the rbeta() in R.

```
bool  MathRandomBeta(
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2
   const int     data_count,     // amount of required data
   double&       result[]        // array to obtain the pseudorandom variables
   );
```

Parameters

a

[in] The first parameter of beta distribution (shape1)

b

[in] The second parameter of beta distribution (shape2).

data_count

[in] The number of pseudorandom variables to be obtained.

error_code

[out] Variable to store the error code.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsBeta

Calculates the theoretical numerical values of the first 4 moments of the beta distribution.

```
double  MathMomentsBeta(
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

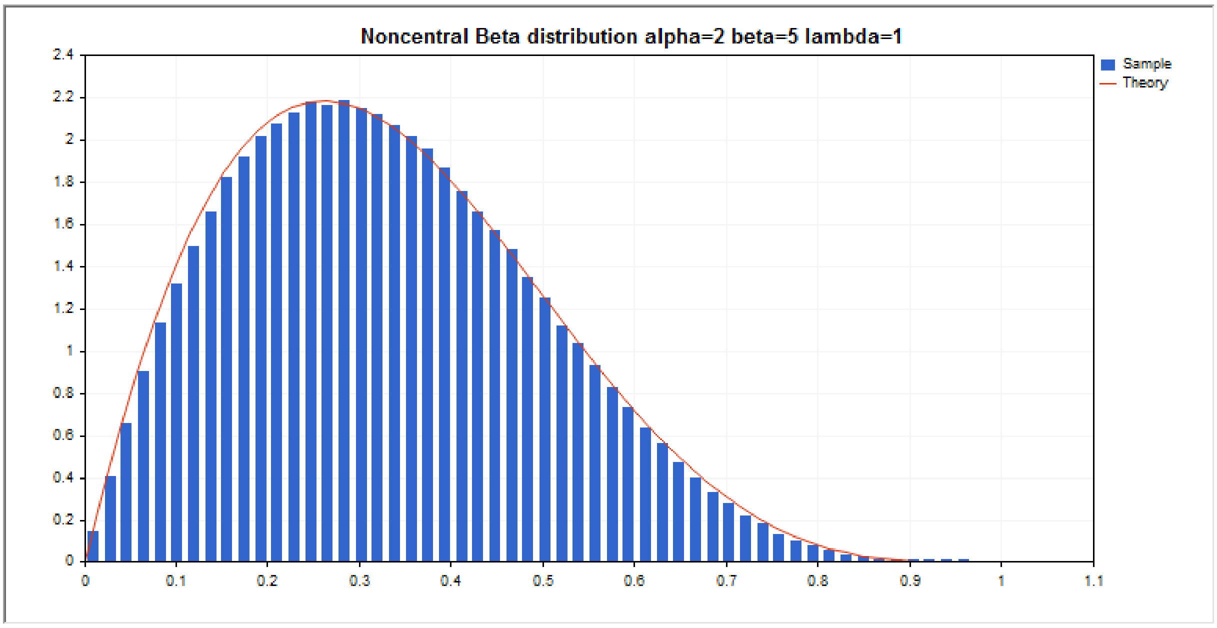
Noncentral beta distribution

This section contains functions for working with noncentral beta distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The noncentral beta distribution is defined by the following formula:

$$f_{NoncentralBeta}(x\,|\,a,b,\lambda)=\sum_{r=0}^{\infty} e^{-\frac{\lambda}{2}}\frac{\left(\frac{\lambda}{2}\right)^{r}}{r!}\,\frac{x^{a+r-1}(1-x)^{b-1}}{B(a+r,b)}$$

where:

- x — value of the random variable
- a — the first parameter of beta distribution
- b — the second parameter of beta distribution
- λ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralBeta	Calculates the probability density function of the noncentral beta distribution
MathCumulativeDistributionNoncentralBeta	Calculates the value of the noncentral beta probability distribution function
MathQuantileNoncentralBeta	Calculates the value of the inverse noncentral beta distribution function for the specified probability
MathRandomNoncentralBeta	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the noncentral beta distribution law

Function	Description
MathMomentsNoncentralBeta	Calculates the theoretical numerical values of the first 4 moments of the noncentral beta distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralBeta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=2;    // the first parameter of beta distribution (shape1)
input double b_par=5;    // the second parameter of beta distribution (shape2)
input double l_par=1;    // noncentrality parameter (lambda)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=53;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral beta distribution
   MathRandomNoncentralBeta(a_par,b_par,l_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralBeta(x2,a_par,b_par,l_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral Beta distribution alpha=%G beta=%G
                          a_par,b_par,l_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                        |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityNoncentralBeta

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const double  lambda,         // noncentrality parameter
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const double  lambda,         // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false. Analog of the dbeta() in R.

```
bool  MathProbabilityDensityNoncentralBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNoncentralBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   double&       result[]         // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

lambda

[in] Noncentrality parameter

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralBeta

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const double  lambda,         // noncentrality parameter
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralBeta(
   const double  x,              // value of random variable
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const double  lambda,         // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false. Analog of the pbeta() in R.

```
bool  MathCumulativeDistributionNoncentralBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the probability distribution function of noncentral beta distribution with the a, b and lambda parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNoncentralBeta(
   const double& x[],             // array with the values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of beta distribution (shape 1).

b

[in] The second parameter of beta distribution (shape 2)

lambda

[in] Noncentrality parameter

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralBeta

Calculates the value of the inverse probability distribution function of noncentral beta distribution with the a, b and lambda parameters for the occurrence probability of a random variable *probability*. In case of error it returns NaN.

```
double  MathQuantileNoncentralBeta(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   int&          error_code       // variable to store the error code
   );
```

Calculates the value of the inverse probability distribution function of noncentral beta distribution with the a, b and lambda parameters for the occurrence probability of a random variable *probability*. In case of error it returns NaN.

```
double  MathQuantileNoncentralBeta(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   int&          error_code       // variable to store the error code
   );
```

For the specified probability[] array of probability values, the function calculates the value of the inverse probability distribution function of noncentral beta distribution with the a, b and lambda parameters. In case of error it returns false. Analog of the qbeta() in R.

```
double  MathQuantileNoncentralBeta(
   const double&  probability[],  // array with probability values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   double&       result[]         // array with values of quantiles
   );
```

For the specified probability[] array of probability values, the function calculates the value of the inverse probability distribution function of noncentral beta distribution with the a, b and lambda parameters. In case of error it returns false.

```
bool  MathQuantileNoncentralBeta(
   const double& probability[],   // array with probability values of random variable
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double   lambda,          // noncentrality parameter
   double&        result[]         // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralBeta

Generates a pseudorandom variable distributed according to the law of noncentral beta distribution the a, b and lambda parameters. In case of error it returns NaN.

```
double  MathRandomNoncentralBeta(
   const double  a,             // the first parameter of beta distribution (shape1)
   const double  b,             // the second parameter of beta distribution (shape2)
   const double  lambda,        // noncentrality parameter
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of noncentral beta distribution the a, b and lambda parameters. In case of error it returns false. Analog of the rbeta() in R.

```
bool  MathRandomNoncentralBeta(
   const double  a,              // the first parameter of beta distribution (shape1)
   const double  b,              // the second parameter of beta distribution (shape2)
   const double  lambda,         // noncentrality parameter
   const int     data_count,     // amount of required data
   double&       result[]        // array to obtain the pseudorandom variables
   );
```

Parameters

a

[in] The first parameter of beta distribution (shape1)

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralBeta

Calculates the theoretical numerical values of the first 4 moments of the noncentral beta distribution with the a, b and lambda parameters.

```
double  MathMomentsNoncentralBeta(
   const double  a,               // the first parameter of beta distribution (shape1)
   const double  b,               // the second parameter of beta distribution (shape2)
   const double  lambda,          // noncentrality parameter
   double&       mean,            // variable for the mean
   double&       variance,        // variable for the variance
   double&       skewness,        // variable for the skewness
   double&       kurtosis,        // variable for the kurtosis
   int&          error_code       // variable for the error code
   );
```

Parameters

a

[in] The first parameter of beta distribution (shape1).

b

[in] The second parameter of beta distribution (shape2).

lambda

[in] Noncentrality parameter

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

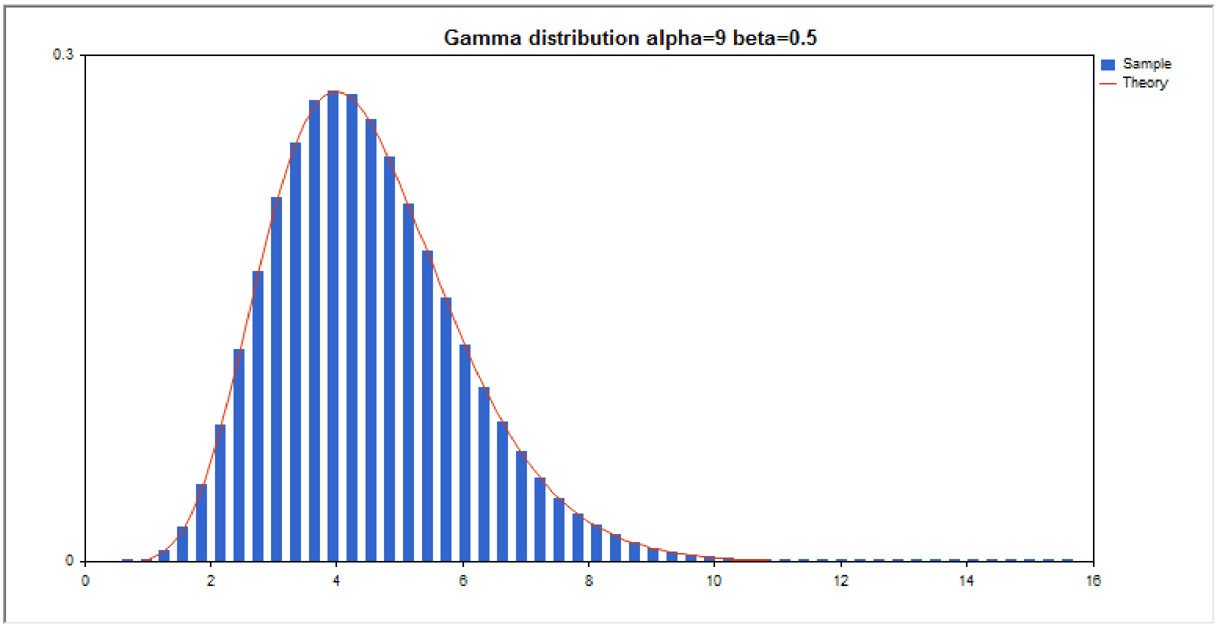
Gamma distribution

This section contains functions for working with gamma distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The gamma distribution is defined by the following formula:

$$f_{Gamma}(x|a,b)=\frac{1}{b^a\Gamma(a)}x^{a-1}e^{-\frac{x}{b}}$$

where:

- x — value of the random variable
- a — the first parameter of distribution
- b — the second parameter of distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityGamma	Calculates the probability density function of the gamma distribution
MathCumulativeDistributionGamma	Calculates the value of the gamma probability distribution function
MathQuantileGamma	Calculates the value of the inverse gamma distribution function for the specified probability
MathRandomGamma	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the gamma distribution law
MathMomentsGamma	Calculates the theoretical numerical values of the first 4 moments of the gamma distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Gamma.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=9;    // the first parameter of distribution (shape)
input double beta=0.5;   // the second parameter of distribution (scale)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the gamma distribution
   MathRandomGamma(alpha,beta,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityGamma(x2,alpha,beta,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Gamma distribution alpha=%G beta=%G",alpha,bet
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(NormalizeDouble(theor_max,1));
   graphic.YAxis().Min(0);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityGamma

Calculates the value of the probability density function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityGamma(
   const double  x,             // value of random variable
   const double  a,             // the first parameter of the distribution (shape)
   const double  b,             // the second parameter of the distribution (scale)
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityGamma(
   const double  x,             // value of random variable
   const double  a,             // the first parameter of the distribution (shape)
   const double  b,             // the second parameter of the distribution (scale)
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the dgamma() in R.

```
bool  MathProbabilityDensityGamma(
   const double& x[],            // array with the values of random variable
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityGamma(
   const double& x[],            // array with the values of random variable
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionGamma

Calculates the probability distribution function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionGamma(
   const double  x,             // value of random variable
   const double  a,             // the first parameter of the distribution (shape)
   const double  b,             // the second parameter of the distribution (scale)
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionGamma(
   const double  x,             // value of random variable
   const double  a,             // the first parameter of the distribution (shape)
   const double  b,             // the second parameter of the distribution (scale)
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the pgamma() in R.

```
bool  MathCumulativeDistributionGamma(
   const double& x[],            // array with the values of random variable
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability function
   );
```

Calculates the probability distribution function of gamma distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionGamma(
   const double& x[],            // array with the values of random variable
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   double&       result[]        // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale)

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileGamma

For the specified *probability*, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileGamma(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of the distribution (shape)
   const double  b,               // the second parameter of the distribution (scale)
   const bool    tail,            // flag of calculation, if false, then calculation i
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculatio
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileGamma(
   const double  probability,     // probability value of random variable occurrence
   const double  a,               // the first parameter of the distribution (shape)
   const double  b,               // the second parameter of the distribution (scale)
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns false. Analog of the qgamma() in R.

```
double  MathQuantileGamma(
   const double& probability[],   // array with probability values of random variable
   const double  a,               // the first parameter of the distribution (shape)
   const double  b,               // the second parameter of the distribution (scale)
   const bool    tail,            // flag of calculation, if false, then calculation i
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculatio
   double&       result[]         // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse gamma distribution function with the a and b parameters. In case of error it returns false.

```
bool  MathQuantileGamma(
   const double& probability[],   // array with probability values of random variable
   const double  a,               // the first parameter of the distribution (shape)
   const double  b,               // the second parameter of the distribution (scale)
   double&       result[]         // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomGamma

Generates a pseudorandom variable distributed according to the law of gamma distribution with the a and b parameters. In case of error it returns NaN.

```
double  MathRandomGamma(
   const double  a,             // the first parameter of the distribution (shape)
   const double  b,             // the second parameter of the distribution (scale)
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of gamma distribution with the a and b parameters. In case of error it returns false. Analog of the rgamma() in R.

```
bool  MathRandomGamma(
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsGamma

Calculates the theoretical numerical values of the first 4 moments of the gamma distribution with the a and b parameters.

```
double  MathMomentsGamma(
   const double  a,              // the first parameter of the distribution (shape)
   const double  b,              // the second parameter of the distribution (scale)
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

a

[in] The first parameter of the distribution (shape).

b

[in] The second parameter of the distribution (scale).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

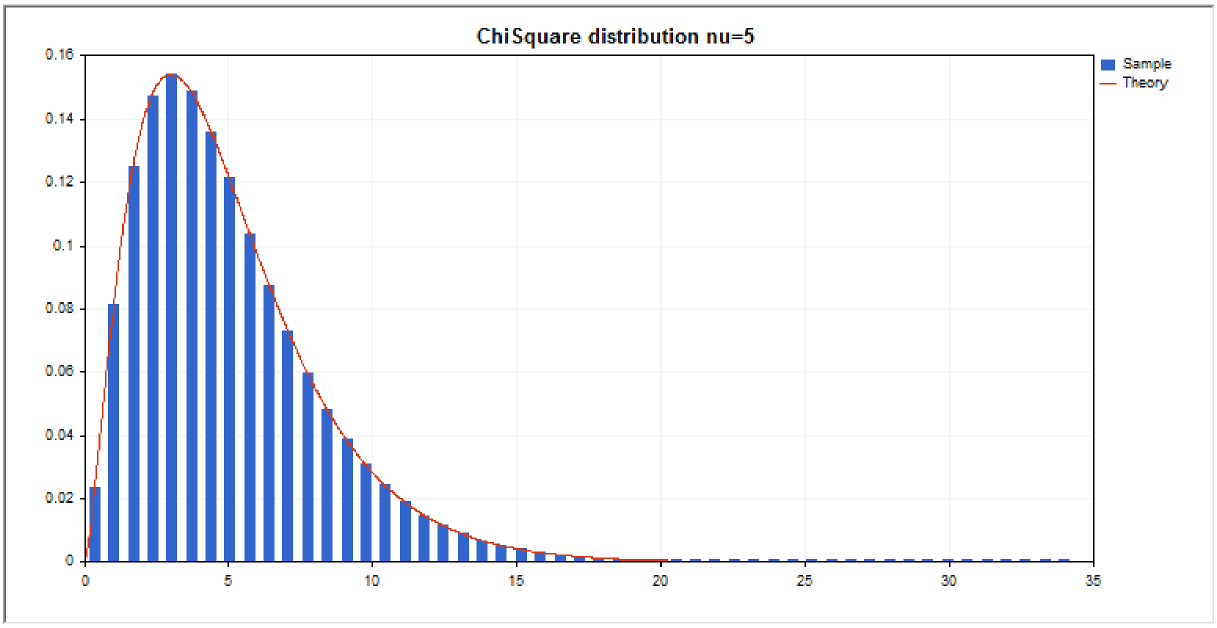
Chi-squared distribution

This section contains functions for working with chi-squared distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The chi-squared distribution is defined by the following formula:

$$f_{Chi-Square}(x\,|\,\nu)=\frac{x^{\frac{(\nu-2)}{2}}e^{-\frac{x}{2}}}{2^{\frac{\nu}{2}}\Gamma\left(\frac{\nu}{2}\right)}$$

where:

- x — value of the random variable
- ν — number of degrees of freedom



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityChiSquare	Calculates the probability density function of the chi-squared distribution
MathCumulativeDistributionChiSquare	Calculates the value of the chi-squared probability distribution function
MathQuantileChiSquare	Calculates the value of the inverse chi-squared distribution function for the specified probability
MathRandomChiSquare	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the chi-squared distribution law

Function	Description
MathMomentsChiSquare	Calculates the theoretical numerical values of the first 4 moments of the chi-squared distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\ChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=5;    // the number of degrees of freedom
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the chi-squared distribution
   MathRandomChiSquare(nu_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityChiSquare(x2,nu_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
```

```
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("ChiSquare distribution nu=%G ",nu_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                        |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityChiSquare

Calculates the value of the probability density function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityChiSquare(
   const double  x,            // value of random variable
   const double  nu,           // parameter of distribution (number of degrees of fre
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=t
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityChiSquare(
   const double  x,            // value of random variable
   const double  nu,           // parameter of distribution (number of degrees of fre
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the dchisq() in R.

```
bool  MathProbabilityDensityChiSquare(
   const double& x[],           // array with the values of random variable
   const double  nu,            // parameter of distribution (number of degrees of fr
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   double&       result[]       // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityChiSquare(
   const double& x[],           // array with the values of random variable
   const double  nu,            // parameter of distribution (number of degrees of fr
   double&       result[]       // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom)

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionChiSquare

Calculates the probability distribution function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionChiSquare(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for a random variable x. In case of error it returns NaN

```
double  MathCumulativeDistributionChiSquare(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the pchisq() in R.

```
bool  MathCumulativeDistributionChiSquare(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the probability distribution function of chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionChiSquare(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileChiSquare

For the specified *probability*, the function calculates the value of inverse chi-squared distribution function. In case of error it returns NaN.

```
double  MathQuantileChiSquare(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of fr
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse chi-squared distribution function. In case of error it returns NaN.

```
double  MathQuantileChiSquare(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of fr
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse chi-squared distribution function. In case of error it returns false. Analog of the qchisq() in R.

```
double  MathQuantileChiSquare(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse chi-squared distribution function. In case of error it returns false.

```
bool  MathQuantileChiSquare(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomChiSquare

Generates a pseudorandom variable distributed according to the law of chi-squared distribution with the nu parameter. In case of error it returns NaN.

```
double  MathRandomChiSquare(
   const double  nu,            // parameter of distribution (number of degrees of fre
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of chi-squared distribution with the nu parameter. In case of error it returns false. Analog of the rchisq() in R.

```
bool  MathRandomChiSquare(
   const double  nu,             // parameter of distribution (number of degrees of fr
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsChiSquare

Calculates the theoretical numerical values of the first 4 moments of the chi-squared distribution with the nu parameter.

```
double  MathMomentsChiSquare(
   const double  nu,           // parameter of distribution (number of degrees of fr
   double&       mean,         // variable for the mean
   double&       variance,     // variable for the variance
   double&       skewness,     // variable for the skewness
   double&       kurtosis,     // variable for the kurtosis
   int&          error_code    // variable for the error code
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

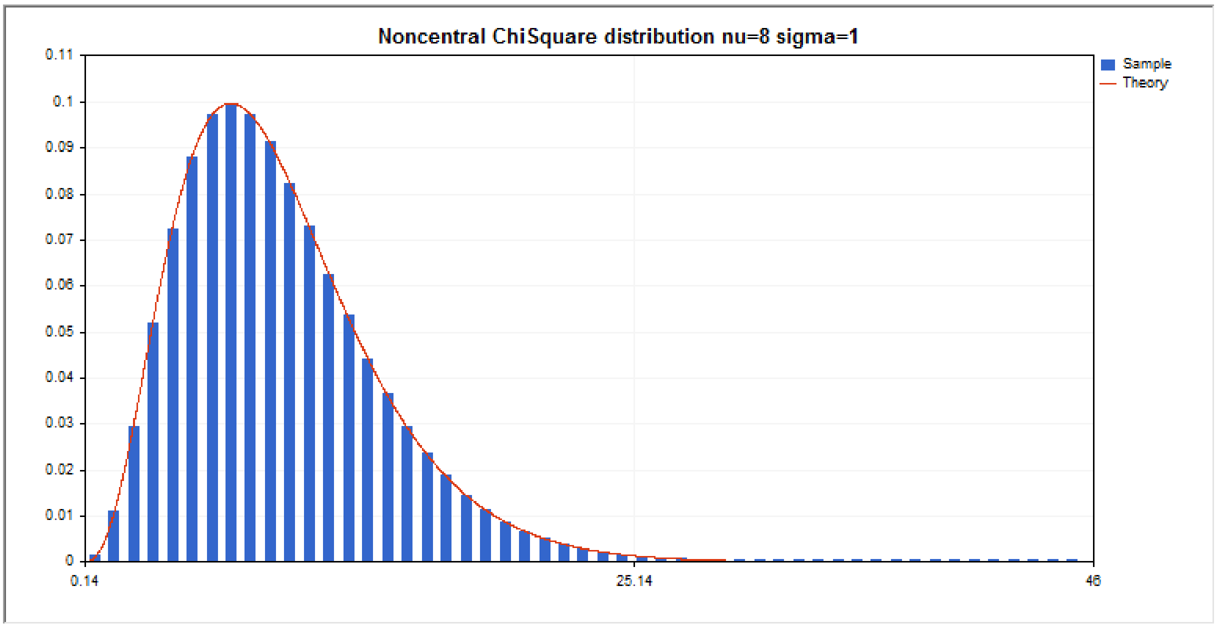
Noncentral chi-squared distribution

This section contains functions for working with noncentral chi-squared distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the corresponding law. The noncentral chi-squared distribution is defined by the following formula:

$$f_{NoncentralChiSquare}(x|\nu,\sigma)=\frac{1}{2^{\frac{\nu}{2}}\Gamma\left(\frac{1}{2}\right)}x^{\frac{\nu}{2}-1}e^{-\frac{(x+\sigma)}{2}}\sum_{r=0}^{\infty}\frac{(\lambda x)^r}{(2r)!}\frac{\Gamma\left(\frac{1}{2}+r\right)}{\Gamma\left(\frac{\nu}{2}+r\right)}$$

where:

- x — value of the random variable
- ν — number of degrees of freedom
- σ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralChiSquare	Calculates the probability density function of the noncentral chi-squared distribution
MathCumulativeDistributionNoncentralChiSquare	Calculates the value of the noncentral chi-squared probability distribution function
MathQuantileNoncentralChiSquare	Calculates the value of the inverse noncentral chi-squared distribution function for the specified probability
MathRandomNoncentralChiSquare	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the noncentral chi-squared distribution law

Function	Description
MathMomentsNoncentralChiSquare	Calculates the theoretical numerical values of the first 4 moments of the noncentral chi-squared distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=8;    // the number of degrees of freedom
input double si_par=1;    // noncentrality parameter
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral chi-squared distribution
   MathRandomNoncentralChiSquare(nu_par,si_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralChiSquare(x2,nu_par,si_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral ChiSquare distribution nu=%G sigma=
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(NormalizeDouble(max,0));
   graphic.XAxis().Min(min);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityNoncentralChiSquare

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralChiSquare(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  sigma,         // noncentrality parameter
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralChiSquare(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  sigma,         // noncentrality parameter
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the dchisq() in R.

```
bool  MathProbabilityDensityNoncentralChiSquare(
   const double& x[],            // array with the values of random variable
   const double  nu,             // parameter of distribution (number of degrees of f
   const double  sigma,          // noncentrality parameter
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of noncentral chi-squared distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNoncentralChiSquare(
   const double& x[],            // array with the values of random variable
   const double  nu,             // parameter of distribution (number of degrees of f
   const double  sigma,          // noncentrality parameter
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralChiSquare

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralChiSquare(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  sigma,         // noncentrality parameter
   const bool    tail,          // flag of calculation, if lower_tail=true, then the p
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralChiSquare(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  sigma,         // noncentrality parameter
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the pchisq() in R.

```
bool  MathCumulativeDistributionNoncentralChiSquare(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const double  sigma,           // noncentrality parameter
   const bool    tail,            // flag of calculation, if lower_tail=true, then the
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the probability distribution function of noncentral chi-squared distribution with the nu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNoncentralChiSquare(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const double  sigma,           // noncentrality parameter
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralChiSquare

For the specified *probability*, the function calculates the value of inverse noncentral chi-squared distribution function with the nu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileNoncentralChiSquare(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of f
   const double  sigma,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if false, then calculation i
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculatio
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse noncentral chi-squared distribution function with the nu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileNoncentralChiSquare(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of f
   const double  sigma,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral chi-squared distribution function with the nu and sigma parameters. In case of error it returns false. Analog of the qchisq() in R.

```
double  MathQuantileNoncentralChiSquare(
   const double&  probability[],  // array with probability values of random variable
   const double  nu,              // parameter of distribution (number of degrees of
   const double  sigma,           // noncentrality parameter
   const bool    tail,            // flag of calculation, if false, then calculation
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculati
   double&       result[]         // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral chi-squared distribution function. In case of error it returns false.

```
bool  MathQuantileNoncentralChiSquare(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of f
   const double  sigma,          // noncentrality parameter
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralChiSquare

Generates a pseudorandom variable distributed according to the law of noncentral chi-squared distribution with the nu and sigma parameters. In case of error it returns NaN.

```
double  MathRandomNoncentralChiSquare(
   const double  nu,             // parameter of distribution (number of degrees of fre
   const double  sigma,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of noncentral chi-squared distribution with the nu and sigma parameters. In case of error it returns false. Analog of the rchisq() in R.

```
bool  MathRandomNoncentralChiSquare(
   const double  nu,              // parameter of distribution (number of degrees of fr
   const double  sigma,           // noncentrality parameter
   const int     data_count,      // amount of required data
   double&       result[]         // array with values of pseudorandom variables
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralChiSquare

Calculates the theoretical numerical values of the first 4 moments of the noncentral chi-squared distribution with the nu and sigma parameters.

```
double  MathMomentsNoncentralChiSquare(
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  sigma,          // noncentrality parameter
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

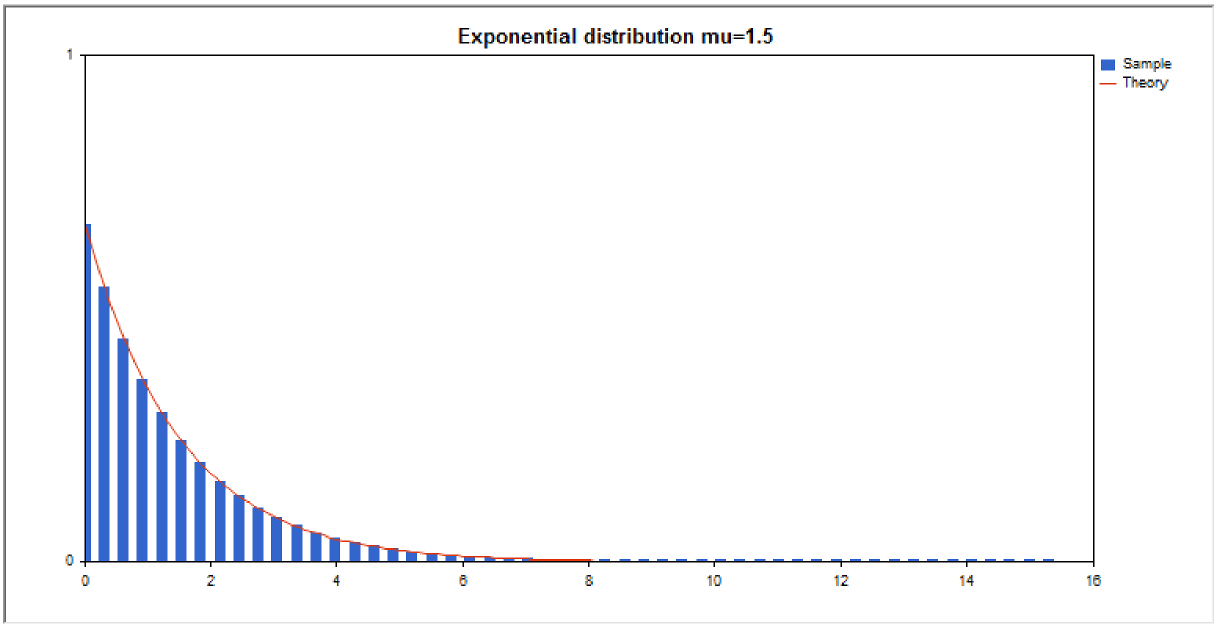
Exponential

This section contains functions for working with exponential distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of exponential distribution. The exponential distribution is defined by the following formula:

$$f_{Exponential}(x|\,\mu) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$$

where:

- x — value of the random variable
- μ — expected value



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityExponential	Calculates the probability density function of the exponential distribution
MathCumulativeDistributionExponential	Calculates the value of the exponential probability distribution function
MathQuantileExponential	Calculates the value of the inverse exponential distribution function for the specified probability
MathRandomExponential	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the law of exponential distribution
MathMomentsExponential	Calculates the theoretical numerical values of the first 4 moments of the exponential distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Exponential.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=1.5;    // the number of degrees of freedom
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the exponential distribution
   MathRandomExponential(mu_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityExponential(x2,mu_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Exponential distribution mu=%G ",mu_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
```

```
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityExponential

Calculates the value of the probability density function of exponential distribution with the mu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityExponential(
   const double  x,             // value of random variable
   const double  mu,            // parameter of the distribution (expected value)
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of exponential distribution with the mu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityExponential(
   const double  x,             // value of random variable
   const double  mu,            // parameter of the distribution (expected value)
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of exponential distribution with the mu parameter for an array of random variables x[]. In case of error it returns false. Analog of the dexp() in R.

```
bool  MathProbabilityDensityExponential(
   const double& x[],            // array with the values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of exponential distribution with the mu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityExponential(
   const double& x[],            // array with the values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   double&       result[]        // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Parameter of the distribution (expected value)

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionExponential

Calculates the exponential distribution function of probabilities with the mu parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionExponential(
   const double  x,             // value of random variable
   const double  mu,            // parameter of the distribution (expected value)
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the exponential distribution function of probabilities with the mu parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionExponential(
   const double  x,             // value of random variable
   const double  mu,            // parameter of the distribution (expected value)
   int&          error_code     // variable to store the error code
   );
```

Calculates the exponential distribution function of probabilities with the mu parameter for an array of random variables x[]. In case of error it returns false. Analog of the pexp() in R.

```
bool  MathCumulativeDistributionExponential(
   const double& x[],            // array with the values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability function
   );
```

Calculates the exponential distribution function of probabilities with the mu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionExponential(
   const double& x[],            // array with the values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   double&       result[]        // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] Parameter of the distribution (expected value).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileExponential

For the specified *probability*, the function calculates the value of inverse exponential distribution function with the mu parameter. In case of error it returns NaN.

```
double  MathQuantileExponential(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // parameter of the distribution (expected value)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse exponential distribution function with the mu parameter. In case of error it returns NaN.

```
double  MathQuantileExponential(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // parameter of the distribution (expected value)
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse exponential distribution function with the mu parameter. In case of error it returns false. Analog of the qexp() in R.

```
double  MathQuantileExponential(
   const double& probability[],  // array with probability values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse exponential distribution function with the mu parameter. In case of error it returns false.

```
bool  MathQuantileExponential(
   const double& probability[],  // array with probability values of random variable
   const double  mu,             // parameter of the distribution (expected value)
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] Parameter of the distribution (expected value).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomExponential

Generates a pseudorandom variable distributed according to the law of exponential distribution with the mu parameter. In case of error it returns NaN.

```
double  MathRandomExponential(
   const double  mu,             // parameter of the distribution (expected value)
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of exponential distribution with the mu parameter. In case of error it returns false. Analog of the rexp() in R.

```
bool  MathRandomExponential(
   const double  mu,              // parameter of the distribution (expected value)
   const int     data_count,      // amount of required data
   double&       result[]         // array with values of pseudorandom variables
   );
```

Parameters

mu

[in] Parameter of the distribution (expected value).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsExponential

Calculates the theoretical numerical values of the first 4 moments of the exponential distribution with the mu parameter.

```
double  MathMomentsExponential(
   const double  mu,             // parameter of the distribution (expected value)
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

mu

[in] Parameter of the distribution (expected value).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

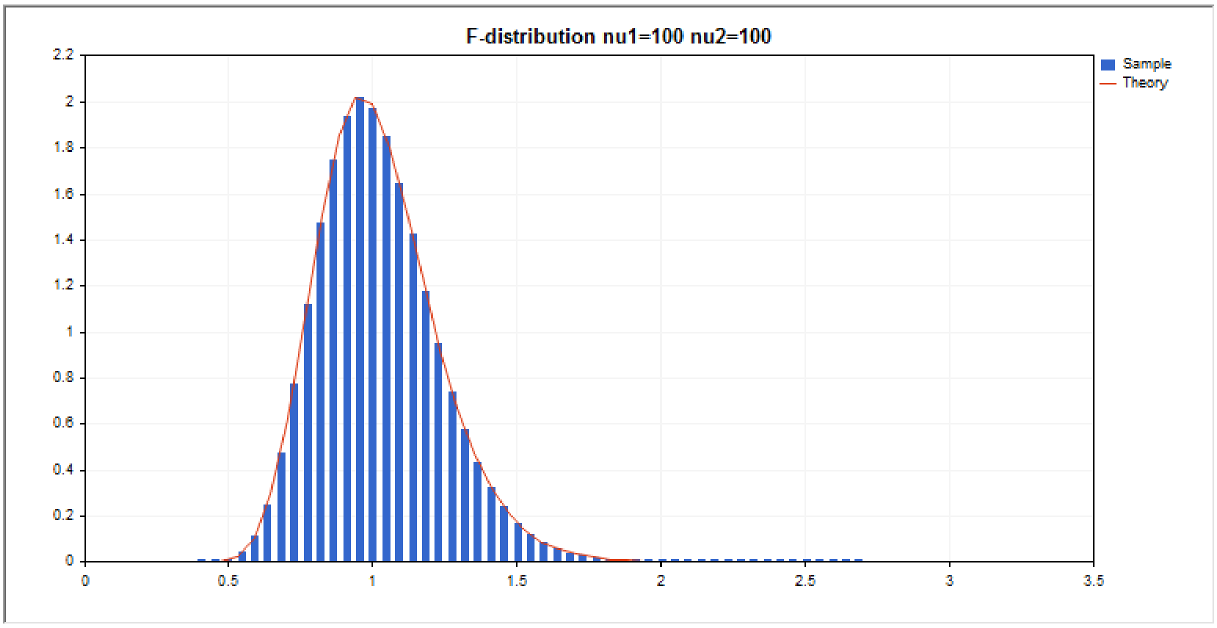
F-distribution

This section contains functions for working with F-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of Fisher's F-distribution. The F-distribution is defined by the following formula:

$$f_F(x|\nu_1,\nu_2)=\frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}\right)}{\Gamma\left(\frac{\nu_1}{2}\right)\Gamma\left(\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}}\frac{x^{\frac{\nu_1-2}{2}}}{\left(1+\left(\frac{\nu_1}{\nu_2}\right)x\right)^{\frac{\nu_1+\nu_2}{2}}}$$

where:

- x — value of the random variable
- ν1 — the first parameter of distribution (number of degrees of freedom)
- ν2 — the second parameter of distribution (number of degrees of freedom)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityF	Calculates the probability density function of the F-distribution
MathCumulativeDistributionF	Calculates the value of the F-distribution function
MathQuantileF	Calculates the value of the inverse F-distribution function for the specified probability
MathRandomF	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Fisher's law

Function	Description
MathMomentsF	Calculates the theoretical numerical values of the first 4 moments of the Fisher's F-distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\F.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=100;    // the first number of degrees of freedom
input double nu_2=100;    // the second number of degrees of freedom
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the Fisher's F-distribution
   MathRandomF(nu_1,nu_2,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityF(x2,nu_1,nu_2,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("F-distribution nu1=%G nu2=%G",nu_1,nu_2));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(4);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

```
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityF

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityF(
   const double  x,            // value of random variable
   const double  nu1,          // the first parameter of distribution (number of degr
   const double  nu2,          // the second parameter of distribution (number of deg
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=t
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityF(
   const double  x,            // value of random variable
   const double  nu1,          // the first parameter of distribution (number of degr
   const double  nu2,          // the second parameter of distribution (number of deg
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false. Analog of the df() in R.

```
bool  MathProbabilityDensityF(
   const double& x[],           // array with the values of random variable
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   double&       result[]       // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityF(
   const double& x[],           // array with the values of random variable
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   double&       result[]       // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionF

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionF(
   const double  x,              // value of random variable
   const double  nu1,            // the first parameter of distribution (number of degr
   const double  nu2,            // the second parameter of distribution (number of deg
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionF(
   const double  x,              // value of random variable
   const double  nu1,            // the first parameter of distribution (number of degr
   const double  nu2,            // the second parameter of distribution (number of deg
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false. Analog of the pf() in R.

```
bool  MathCumulativeDistributionF(
   const double& x[],              // array with the values of random variable
   const double  nu1,              // the first parameter of distribution (number of deg
   const double  nu2,              // the second parameter of distribution (number of de
   const bool    tail,             // flag of calculation, if true, then the probability
   const bool    log_mode,         // calculate the logarithm of the value, if log_mode=
   double&       result[]          // array for values of the probability function
   );
```

Calculates the value of the probability distribution function of Fisher's F-distribution with the nu1 and nu2 parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionF(
   const double& x[],              // array with the values of random variable
   const double  nu1,              // the first parameter of distribution (number of deg
   const double  nu2,              // the second parameter of distribution (number of de
   double&       result[]          // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileF

For the specified *probability*, the function calculates the value of inverse Fisher's F-distribution function with the nu1 and nu2 parameters. In case of error it returns NaN.

```
double  MathQuantileF(
   const double  probability,    // probability value of random variable occurrence
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse Fisher's F-distribution function with the nu1 and nu2 parameters. In case of error it returns NaN.

```
double  MathQuantileF(
   const double  probability,    // probability value of random variable occurrence
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Fisher's F-distribution function with the nu1 and nu2 parameters. In case of error it returns false. Analog of the qf() in R.

```
double  MathQuantileF(
   const double& probability[],  // array with probability values of random variable
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Fisher's F-distribution function with the nu1 and nu2 parameters. In case of error it returns false.

```
bool  MathQuantileF(
   const double& probability[],  // array with probability values of random variable
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

`probability[]`

[in] Array with probability values of random variable.

`nu1`

[in] The first parameter of distribution (number of degrees of freedom).

`nu2`

[in] The second parameter of distribution (number of degrees of freedom).

`tail`

[in] Flag of calculation, if lower_tail=false, then calculation is performed for 1.0-probability.

`log_mode`

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

`error_code`

[out] Variable to get the error code.

`result[]`

[out] Array with values of quantiles.

MathRandomF

Generates a pseudorandom variable distributed according to the law of Fisher's F-distribution with the nu1 and nu2 parameters. In case of error it returns NaN.

```
double  MathRandomF(
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of Fisher's F-distribution with the nu1 and nu2 parameters. In case of error it returns false. Analog of the rf() in R.

```
bool  MathRandomF(
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   const int     data_count,    // amount of required data
   double&       result[]       // array with values of pseudorandom variables
   );
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsF

Calculates the theoretical numerical values of the first 4 moments of the Fisher's F-distribution with the nu1 and nu2 parameters.

```
double  MathMomentsF(
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   double&       mean,          // variable for the mean
   double&       variance,      // variable for the variance
   double&       skewness,      // variable for the skewness
   double&       kurtosis,      // variable for the kurtosis
   int&          error_code     // variable for the error code
   );
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

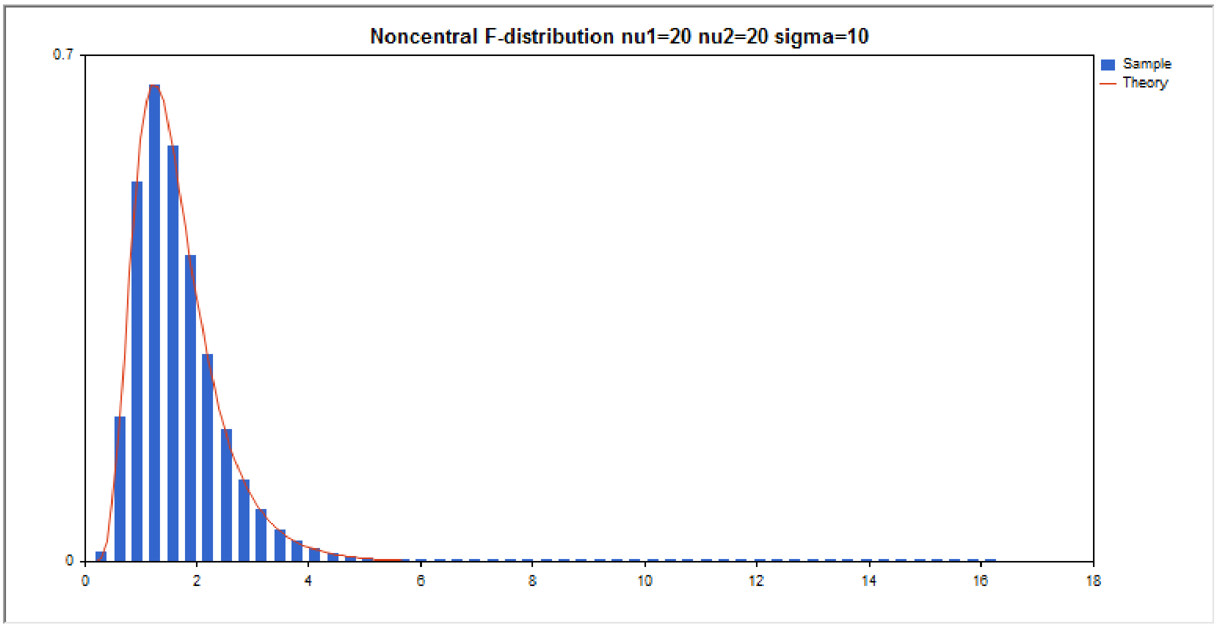
Noncentral F-distribution

This section contains functions for working with noncentral F-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of noncentral Fisher's F-distribution. The noncentral F-distribution is defined by the following formula:

$$f_{NoncentralF}(x \mid \nu_1;\nu_2;\sigma)=e^{-\frac{\sigma}{2}}\sum_{r=0}^{\infty}\frac{1}{r!}\left(\frac{\sigma}{2}\right)^{r}\frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}+r\right)}{\Gamma\left(\frac{\nu_2}{2}+r\right)\Gamma\left(\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_2}{2}+r}\frac{x^{\frac{\nu_2}{2}-1+r}}{\left(1+\frac{\nu_1}{\nu_2}x\right)^{\frac{\nu_1+\nu_2}{2}+r}}$$

where:

- x — value of the random variable
- ν1 — the first parameter of distribution (number of degrees of freedom)
- ν2 — the second parameter of distribution (number of degrees of freedom)
- σ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralF	Calculates the probability density function of the noncentral F-distribution
MathCumulativeDistributionNoncentralF	Calculates the value of the noncentral F-distribution function
MathQuantileNoncentralF	Calculates the value of the inverse noncentral F-distribution function for the specified probability

Function	Description
MathRandomNoncentralF	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the noncentral Fisher's distribution law
MathMomentsNoncentralF	Calculates the theoretical numerical values of the first 4 moments of the noncentral Fisher's F-distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralF.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=20;    // the first number of degrees of freedom
input double nu_2=20;    // the second number of degrees of freedom
input double sig=10;     // noncentrality parameter
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral Fisher's F-distribution
   MathRandomNoncentralF(nu_1,nu_2,sig,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralF(x2,nu_1,nu_2,sig,false,y2);
```

```
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral F-distribution nu1=%G nu2=%G sigma=
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityNoncentralF

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralF(
   const double  x,             // value of random variable
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   const double  sigma,         // noncentrality parameter
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralF(
   const double  x,             // value of random variable
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   const double  sigma,         // noncentrality parameter
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the df() in R.

```
bool  MathProbabilityDensityNoncentralF(
   const double& x[],            // array with the values of random variable
   const double  nu1,            // the first parameter of distribution (number of de
   const double  nu2,            // the second parameter of distribution (number of d
   const double  sigma,          // noncentrality parameter
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNoncentralF(
   const double& x[],            // array with the values of random variable
   const double  nu1,            // the first parameter of distribution (number of de
   const double  nu2,            // the second parameter of distribution (number of d
   const double  sigma,          // noncentrality parameter
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralF

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralF(
   const double  x,              // value of random variable
   const double  nu1,            // the first parameter of distribution (number of degr
   const double  nu2,            // the second parameter of distribution (number of deg
   const double  sigma,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if lo
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralF(
   const double  x,              // value of random variable
   const double  nu1,            // the first parameter of distribution (number of degr
   const double  nu2,            // the second parameter of distribution (number of deg
   const double  sigma,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the pf() in R.

```
bool  MathCumulativeDistributionNoncentralF(
   const double& x[],             // array with the values of random variable
   const double  nu1,             // the first parameter of distribution (number of deg
   const double  nu2,             // the second parameter of distribution (number of de
   const double  sigma,           // noncentrality parameter
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the probability distribution function of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNoncentralF(
   const double& x[],             // array with the values of random variable
   const double  nu1,             // the first parameter of distribution (number of deg
   const double  nu2,             // the second parameter of distribution (number of de
   const double  sigma,           // noncentrality parameter
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileF

For the specified *probability*, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileF(
   const double  probability,    // probability value of random variable occurrence
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   const double  sigma,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileF(
   const double  probability,    // probability value of random variable occurrence
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   const double  sigma,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns false. Analog of the qf() in R.

```
double  MathQuantileF(
   const double& probability[],  // array with probability values of random variable
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   const double  sigma,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Fisher's F-distribution function with the nu1, nu2 and sigma parameters. In case of error it returns false.

```
bool  MathQuantileF(
   const double& probability[],  // array with probability values of random variable
   const double  nu1,            // the first parameter of distribution (number of deg
   const double  nu2,            // the second parameter of distribution (number of de
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralF

Generates a pseudorandom variable distributed according to the law of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters. In case of error it returns NaN.

```
double  MathRandomNoncentralF(
   const double  nu1,             // the first parameter of distribution (number of deg
   const double  nu2,             // the second parameter of distribution (number of de
   const double  sigma,           // noncentrality parameter
   int&          error_code       // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters. In case of error it returns false. Analog of the rf() in R.

```
bool  MathRandomNoncentralF(
   const double  nu1,             // the first parameter of distribution (number of deg
   const double  nu2,             // the second parameter of distribution (number of de
   const double  sigma,           // noncentrality parameter
   const int     data_count,      // amount of required data
   double&       result[]         // array with values of pseudorandom variables
   );
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralF

Calculates the theoretical numerical values of the first 4 moments of the noncentral Fisher's F-distribution with the nu1, nu2 and sigma parameters.

```
double  MathMomentsNoncentralF(
   const double  nu1,           // the first parameter of distribution (number of deg
   const double  nu2,           // the second parameter of distribution (number of de
   const double  sigma,         // noncentrality parameter
   double&       mean,          // variable for the mean
   double&       variance,      // variable for the variance
   double&       skewness,      // variable for the skewness
   double&       kurtosis,      // variable for the kurtosis
   int&          error_code     // variable for the error code
   );
```

Parameters

nu1

[in] The first parameter of distribution (number of degrees of freedom).

nu2

[in] The second parameter of distribution (number of degrees of freedom).

sigma

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

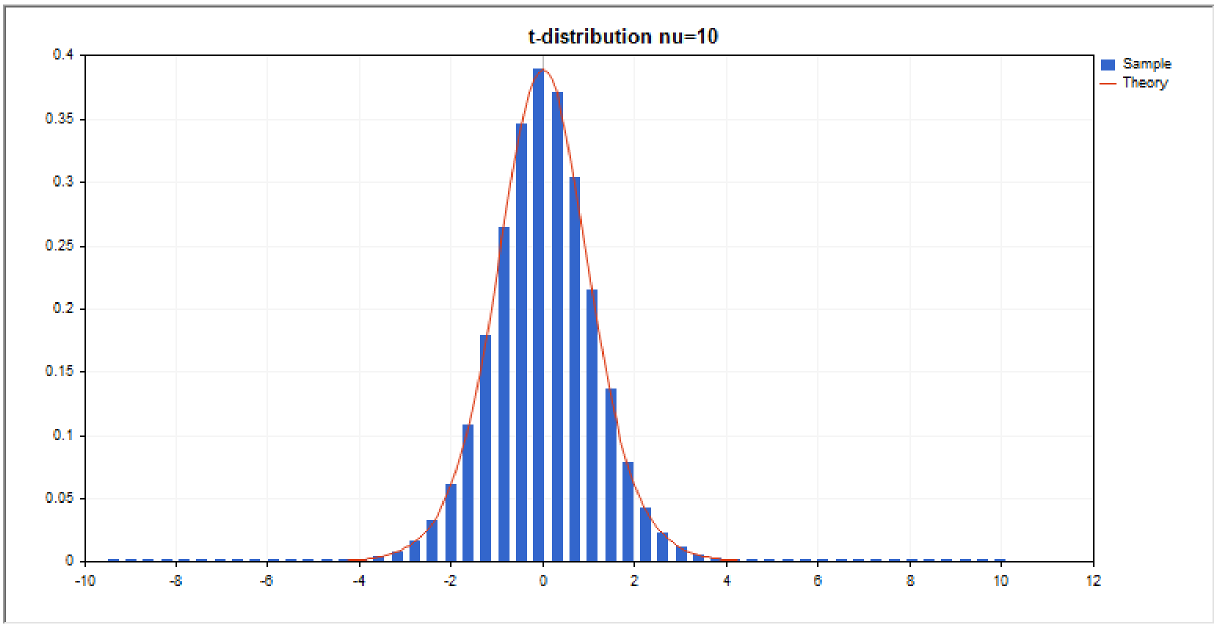
t-distribution

This section contains functions for working with Student's t-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Student's law. The t-distribution is defined by the following formula:

$$f_T(x|\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\pi\nu}} \frac{1}{\left(1+\frac{x^2}{\nu}\right)^{\frac{\nu+1}{2}}}$$

where:

- x — value of the random variable
- ν — parameter of distribution (number of degrees of freedom)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityT	Calculates the probability density function of the t-distribution
MathCumulativeDistributionT	Calculates the value of the t-distribution function
MathQuantileT	Calculates the value of the inverse t-distribution function for the specified probability
MathRandomT	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the law of Student's t-distribution

Function	Description
MathMomentsT	Calculates the theoretical numerical values of the first 4 moments of the Student's t-distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\T.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=10;    // the number of degrees of freedom
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the Student's t-distribution
   MathRandomT(nu_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityT(x2,nu_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
```

```
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("t-distribution nu=%G",nu_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityT

Calculates the value of the probability density function of Student's t-distribution with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityT(
   const double  x,            // value of random variable
   const double  nu,           // parameter of distribution (number of degrees of fre
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=t
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Student's t-distribution with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityT(
   const double  x,            // value of random variable
   const double  nu,           // parameter of distribution (number of degrees of fre
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Student's t-distribution with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the dt() in R.

```
bool  MathProbabilityDensityT(
   const double& x[],           // array with the values of random variable
   const double  nu,            // parameter of distribution (number of degrees of fr
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   double&       result[]       // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of Student's t-distribution with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityT(
   const double& x[],           // array with the values of random variable
   const double  nu,            // parameter of distribution (number of degrees of fr
   double&       result[]       // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionT

Calculates the value of the Student's t-distribution function with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionT(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if lo
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Student's t-distribution function with the nu parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionT(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Student's t-distribution function with the nu parameter for an array of random variables x[]. In case of error it returns false. Analog of the pt() in R.

```
bool  MathCumulativeDistributionT(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the Student's t-distribution function with the nu parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionT(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileT

For the specified *probability*, the function calculates the value of inverse Student's t-distribution function with the nu parameter. In case of error it returns NaN.

```
double  MathQuantileT(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of f
   const bool    tail,           // flag of calculation, if false, then calculation i
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculatio
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse Student's t-distribution function with the nu parameter. In case of error it returns NaN.

```
double  MathQuantileT(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of f
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Student's t-distribution function with the nu parameter. In case of error it returns false. Analog of the qt() in R.

```
double  MathQuantileT(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of f
   const bool    tail,           // flag of calculation, if false, then calculation i
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculatio
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Student's t-distribution function with the nu parameter. In case of error it returns false.

```
bool  MathQuantileT(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of f
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomT

Generates a pseudorandom variable distributed according to the law of Student's t-distribution with the nu parameter. In case of error it returns NaN.

```
double  MathRandomT(
   const double  nu,              // parameter of distribution (number of degrees of fr
   int&          error_code       // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of Student's t-distribution with the nu parameter. In case of error it returns false. Analog of the rt() in R.

```
bool  MathRandomT(
   const double  nu,              // parameter of distribution (number of degrees of fr
   const int     data_count,      // amount of required data
   double&       result[]         // array with values of pseudorandom variables
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsT

Calculates the theoretical numerical values of the first 4 moments of the Student's t-distribution with the nu parameter.

```
double  MathMomentsT(
   const double  nu,            // parameter of distribution (number of degrees of fr
   double&       mean,          // variable for the mean
   double&       variance,      // variable for the variance
   double&       skewness,      // variable for the skewness
   double&       kurtosis,      // variable for the kurtosis
   int&          error_code     // variable for the error code
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

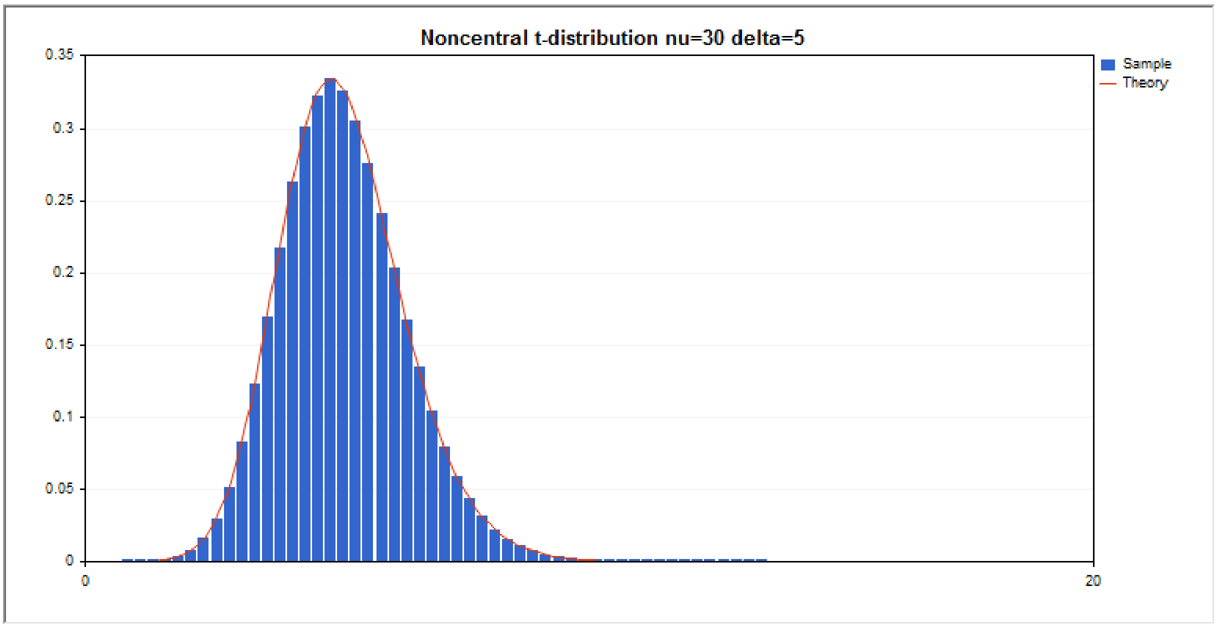
t-distribution

This section contains functions for working with noncentral Student's t-distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the law of noncentral t-distribution. The noncentral t-distribution is defined by the following formula:

$$f_{NoncentralT}(x \mid \nu,\delta) = \frac{\nu^{\frac{\nu}{2}} e^{-\frac{\sigma^2}{2}}}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi}\left(\nu+x^2\right)^{\frac{\nu+1}{2}}} \sum_{r=0}^{\infty} \frac{(x\delta)^r}{r!} \left(\frac{2}{\nu+x^2}\right)^{\frac{r}{2}} \Gamma\left(\frac{\nu+r+1}{2}\right)$$

where:

- x — value of the random variable
- ν — parameter of distribution (number of degrees of freedom)
- σ — noncentrality parameter



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNoncentralT	Calculates the probability density function of the noncentral t-distribution
MathCumulativeDistributionNoncentralT	Calculates the value of the noncentral t-distribution function
MathQuantileNoncentralT	Calculates the value of the inverse noncentral t-distribution function for the specified probability
MathRandomNoncentralT	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the noncentral Student's t-distribution law

Function	Description
MathMomentsNoncentralT	Calculates the theoretical numerical values of the first 4 moments of the noncentral Student's t-distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralT.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=30;       // the number of degrees of freedom
input double delta_par=5;    // noncentrality parameter
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the noncentral Student's t-distribution
   MathRandomNoncentralT(nu_par,delta_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityNoncentralT(x2,nu_par,delta_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Noncentral t-distribution nu=%G delta=%G",nu_
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

```
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityNoncentralT

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralT(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   const double  delta,          // noncentrality parameter
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable for the error code
   );
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNoncentralT(
   const double  x,              // value of random variable
   const double  nu,             // parameter of distribution (number of degrees of fre
   const double  delta,          // noncentrality parameter
   int&          error_code      // variable for the error code
   );
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false. Analog of the dt() in R.

```
bool  MathProbabilityDensityNoncentralT(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const double  delta,           // noncentrality parameter
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNoncentralT(
   const double& x[],             // array with the values of random variable
   const double  nu,              // parameter of distribution (number of degrees of fr
   const double  delta,           // noncentrality parameter
   double&       result[]         // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNoncentralT

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralT(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  delta,         // noncentrality parameter
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // flag to calculate the logarithm of the value, if lo
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNoncentralT(
   const double  x,             // value of random variable
   const double  nu,            // parameter of distribution (number of degrees of fre
   const double  delta,         // noncentrality parameter
   int&          error_code     // variable to store the error code
   );
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false. Analog of the pt() in R.

```
bool  MathCumulativeDistributionNoncentralT(
   const double& x[],            // array with the values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability function
   );
```

Calculates the probability distribution function of noncentral Student's t-distribution with the nu and delta parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNoncentralT(
   const double& x[],            // array with the values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   double&       result[]        // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNoncentralT

For the specified *probability*, the function calculates the value of inverse noncentral Student's t-distribution function with the nu and delta parameters. In case of error it returns NaN.

```
double  MathQuantileNoncentralT(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if lower_tail=false, then cal
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse noncentral Student's t-distribution function with the nu and delta parameters. In case of error it returns NaN.

```
double  MathQuantileNoncentralT(
   const double  probability,    // probability value of random variable occurrence
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Student's t-distribution function with the nu and delta parameters. In case of error it returns false. Analog of the qt() in R.

```
double  MathQuantileNoncentralT(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   const bool    tail,           // flag of calculation, if lower_tail=false, then cal
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse noncentral Student's t-distribution function with the nu and delta parameters. In case of error it returns false.

```
bool  MathQuantileNoncentralT(
   const double& probability[],  // array with probability values of random variable
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNoncentralT

Generates a pseudorandom variable distributed according to the law of noncentral Student's t-distribution with the nu and delta parameters. In case of error it returns NaN.

```
double  MathRandomNoncentralT(
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of noncentral Student's t-distribution with the nu and delta parameters. In case of error it returns false. Analog of the rt() in R.

```
bool  MathRandomNoncentralT(
   const double  nu,             // parameter of distribution (number of degrees of fr
   const double  delta,          // noncentrality parameter
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNoncentralT

Calculates the theoretical numerical values of the first 4 moments of the noncentral Student's t-distribution with the nu and delta parameters.

```
double  MathMomentsNoncentralT(
   const double  nu,            // parameter of distribution (number of degrees of fr
   const double  delta,         // noncentrality parameter
   double&       mean,          // variable for the mean
   double&       variance,      // variable for the variance
   double&       skewness,      // variable for the skewness
   double&       kurtosis,      // variable for the kurtosis
   int&          error_code     // variable for the error code
   );
```

Parameters

nu

[in] Parameter of distribution (number of degrees of freedom).

delta

[in] Noncentrality parameter.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

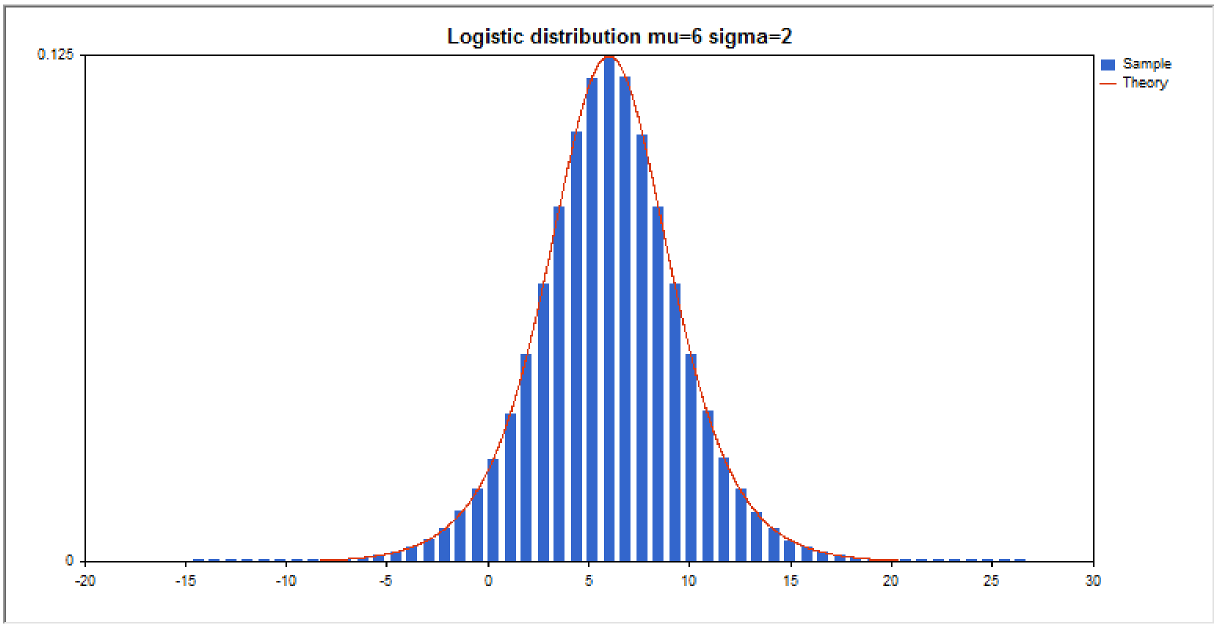
Logistic distribution

This section contains functions for working with logistic distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the logistic law. The logistic distribution is defined by the following formula:

$$f_{Logistic}(x|\mu,\sigma)=\frac{e^{\frac{x-\mu}{\sigma}}}{\sigma\left(1+e^{\frac{x-\mu}{\sigma}}\right)^2}$$

where:

- x — value of the random variable
- µ — mean parameter of the distribution
- σ — scale parameter of the distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityLogistic	Calculates the probability density function of the logistic distribution
MathCumulativeDistributionLogistic	Calculates the value of the logistic probability distribution function
MathQuantileLogistic	Calculates the value of the inverse logistic distribution function for the specified probability
MathRandomLogistic	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the logistic law

Function	Description
MathMomentsLogistic	Calculates the theoretical numerical values of the first 4 moments of the logistic distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Logistic.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=6;        // mean parameter of the distribution
input double sigma_par=2;     // scale parameter of the distribution
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the logistic distribution
   MathRandomLogistic(mu_par,sigma_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityLogistic(x2,mu_par,sigma_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Logistic distribution mu=%G sigma=%G",mu_par,s
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(theor_max);
   graphic.YAxis().Min(0);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityLogistic

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityLogistic(
   const double  x,             // value of random variable
   const double  mu,            // mean parameter of the distribution
   const double  sigma,         // scale parameter of the distribution
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityLogistic(
   const double  x,             // value of random variable
   const double  mu,            // mean parameter of the distribution
   const double  sigma,         // scale parameter of the distribution
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the dlogis() in R.

```
bool  MathProbabilityDensityLogistic(
   const double& x[],            // array with the values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of logistic distribution with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityLogistic(
   const double& x[],            // array with the values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   double&       result[]        // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionLogistic

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionLogistic(
   const double  x,             // value of random variable
   const double  mu,            // mean parameter of the distribution
   const double  sigma,         // scale parameter of the distribution
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // flag to calculate the logarithm of the value, if lo
   int&          error_code     // variable to store the error code
   );
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionLogistic(
   const double  x,             // value of random variable
   const double  mu,            // mean parameter of the distribution
   const double  sigma,         // scale parameter of the distribution
   int&          error_code     // variable to store the error code
   );
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionLogistic(
   const double& x[],            // array with the values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability function
   );
```

Calculates the logistic distribution function of probabilities with the mu and sigma parameters for an array of random variables x[]. In case of error it returns false. Analog of the plogis() in R.

```
bool  MathCumulativeDistributionLogistic(
   const double& x[],            // array with the values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   double&       result[]        // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileLogistic

For the specified *probability*, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileLogistic(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   const bool    tail,           // flag of calculation, if false, then calculation i
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculatio
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathQuantileLogistic(
   const double  probability,    // probability value of random variable occurrence
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns false. Analog of the qlogis() in R.

```
double  MathQuantileLogistic(
   const double& probability[],  // array with probability values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   const bool    tail,           // flag of calculation, if false, then calculation i
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculatio
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse logistic distribution function with the mu and sigma parameters. In case of error it returns false.

```
bool  MathQuantileLogistic(
   const double& probability[],  // array with probability values of random variable
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomLogistic

Generates a pseudorandom variable distributed according to the law of logistic distribution with the mu and sigma parameters. In case of error it returns NaN.

```
double  MathRandomLogistic(
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of logistic distribution with the mu and sigma parameters. In case of error it returns false. Analog of the rlogis() in R.

```
bool  MathRandomLogistic(
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsLogistic

Calculates the theoretical numerical values of the first 4 moments of the logistic distribution with the mu and sigma parameters.

```
double  MathMomentsLogistic(
   const double  mu,             // mean parameter of the distribution
   const double  sigma,          // scale parameter of the distribution
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

mu

[in] mean parameter of the distribution.

sigma

[in] scale parameter of the distribution.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

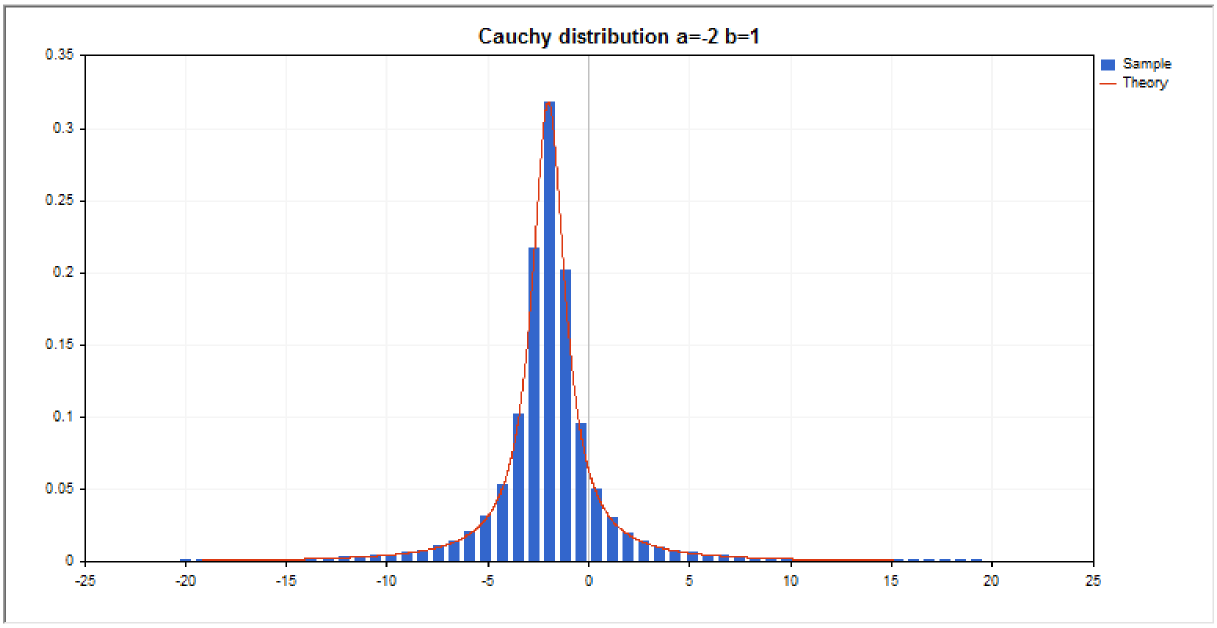
Cauchy distribution

This section contains functions for working with Cauchy distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Cauchy law. The Cauchy distribution is defined by the following formula:

$$f_{Cauchy}(x|a,b)=\frac{1}{\pi}\frac{b}{\left(b^2+(x-a)^2\right)}$$

where:

- x — value of the random variable
- a — mean parameter of the distribution
- b — scale parameter of the distribution



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityCauchy	Calculates the probability density function of the Cauchy distribution
MathCumulativeDistributionCauchy	Calculates the value of the Cauchy probability distribution function
MathQuantileCauchy	Calculates the value of the inverse Cauchy distribution function for the specified probability
MathRandomCauchy	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Cauchy law
MathMomentsCauchy	Calculates the theoretical numerical values of the first 4 moments of the Cauchy distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Cauchy.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=-2;      // mean parameter of the distribution
input double b_par=1;       // scale parameter of the distribution
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the Cauchy distribution
   MathRandomCauchy(a_par,b_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityCauchy(x2,a_par,b_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Cauchy distribution a=%G b=%G",a_par,b_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1)
      return(false);
   int size=ArraySize(data);
   if(size<cells*10)
      return(false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   Print("min=",minv," max=",maxv);
   minv=-20;
   maxv=20;
   double range=maxv-minv;
   double width=range/cells;
   if(width==0)
      return(false);
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=(int)MathRound((data[i]-minv)/width);
      if(ind>=0 && ind<cells)
         frequency[ind]++;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityCauchy

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityCauchy(
   const double  x,            // value of random variable
   const double  a,            // mean parameter of the distribution
   const double  b,            // scale parameter of the distribution
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=t
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityCauchy(
   const double  x,            // value of random variable
   const double  a,            // mean parameter of the distribution
   const double  b,            // scale parameter of the distribution
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the dcauchy() in R.

```
bool  MathProbabilityDensityCauchy(
   const double& x[],           // array with the values of random variable
   const double  a,             // mean parameter of the distribution
   const double  b,             // scale parameter of the distribution
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   double&       result[]       // array for values of the probability density functi
   );
```

Calculates the value of the probability density function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityCauchy(
   const double& x[],           // array with the values of random variable
   const double  a,             // mean parameter of the distribution
   const double  b,             // scale parameter of the distribution
   double&       result[]       // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionCauchy

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionCauchy(
   const double  x,           // value of random variable
   const double  a,           // mean parameter of the distribution
   const double  b,           // scale parameter of the distribution
   const bool    tail,        // flag of calculation, if true, then the probability
   const bool    log_mode,    // flag to calculate the logarithm of the value, if lo
   int&          error_code   // variable to store the error code
   );
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionCauchy(
   const double  x,           // value of random variable
   const double  a,           // mean parameter of the distribution
   const double  b,           // scale parameter of the distribution
   int&          error_code   // variable to store the error code
   );
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionCauchy(
   const double& x[],          // array with the values of random variable
   const double  a,            // mean parameter of the distribution
   const double  b,            // scale parameter of the distribution
   const bool    tail,         // flag of calculation, if true, then the probability
   const bool    log_mode,     // flag to calculate the logarithm of the value, if l
   double&       result[]      // array for values of the probability function
   );
```

Calculates the probability distribution function of Cauchy distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the plogis() in R.

```
bool  MathCumulativeDistributionCauchy(
   const double& x[],          // array with the values of random variable
   const double  a,            // mean parameter of the distribution
   const double  b,            // scale parameter of the distribution
   double&       result[]      // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileCauchy

For the specified *probability*, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileCauchy(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileCauchy(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns false. Analog of the qcauschy() in R.

```
double  MathQuantileCauchy(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Cauchy distribution function with the a and b parameters. In case of error it returns false.

```
bool  MathQuantileCauchy(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomCauchy

Generates a pseudorandom variable distributed according to the law of Cauchy distribution with the a and b parameters. In case of error it returns NaN.

```
double  MathRandomCauchy(
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of Cauchy distribution with the a and b parameters. In case of error it returns false. Analog of the rcauchy() in R.

```
bool  MathRandomCauchy(
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsCauchy

Calculates the theoretical numerical values of the first 4 moments of the Cauchy distribution with the a and b parameters.

```
double  MathMomentsCauchy(
   const double  a,              // mean parameter of the distribution
   const double  b,              // scale parameter of the distribution
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

a

[in] mean parameter of the distribution.

b

[in] scale parameter of the distribution.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

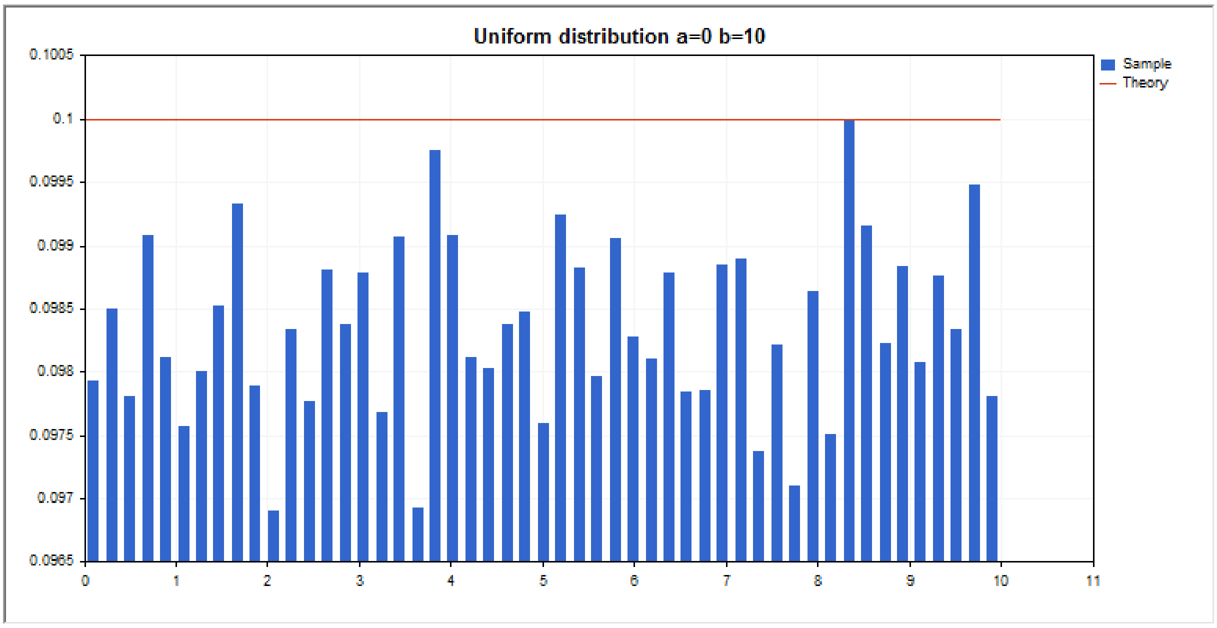
Uniform distribution

This section contains functions for working with uniform distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the uniform law. The uniform distribution is defined by the following formula:

$$f_{Uniform}(x|\,a,b)=\frac{1}{b-a}$$

where:

- x — value of the random variable
- a — parameter of the distribution (lower bound)
- b — parameter of the distribution (upper bound)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityUniform	Calculates the probability density function of the uniform distribution
MathCumulativeDistributionUniform	Calculates the value of the uniform probability distribution function
MathQuantileUniform	Calculates the value of the inverse uniform distribution function for the specified probability
MathRandomUniform	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the uniform law
MathMomentsUniform	Calculates the theoretical numerical values of the first 4 moments of the uniform distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Uniform.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=0;      // distribution parameter a (lower bound)
input double b_par=10;     // distribution parameter b (upper bound)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the uniform distribution
   MathRandomUniform(a_par,b_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityUniform(x2,a_par,b_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Uniform distribution a=%G b=%G",a_par,b_par));
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
```

```
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityUniform

Calculates the value of the probability density function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityUniform(
   const double  x,             // value of random variable
   const double  a,             // distribution parameter a (lower bound)
   const double  b,             // distribution parameter b (upper bound)
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityUniform(
   const double  x,             // value of random variable
   const double  a,             // distribution parameter a (lower bound)
   const double  b,             // distribution parameter b (upper bound)
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the dunif() in R.

```
bool  MathProbabilityDensityUniform(
   const double& x[],            // array with the values of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityUniform(
   const double& x[],            // array with the values of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionUniform

Calculates the probability distribution function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionUniform(
   const double  x,              // value of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionUniform(
   const double  x,              // value of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   int&          error_code      // variable to store the error code
   );
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionUniform(
   const double& x[],             // array with the values of random variable
   const double  a,               // distribution parameter a (lower bound)
   const double  b,               // distribution parameter b (upper bound)
   const bool    tail,            // flag of calculation, if true, then the probabilit
   const bool    log_mode,        // flag to calculate the logarithm of the value, if
   double&       result[]         // array for values of the probability function
   );
```

Calculates the probability distribution function of uniform distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the punif() in R.

```
bool  MathCumulativeDistributionUniform(
   const double& x[],             // array with the values of random variable
   const double  a,               // distribution parameter a (lower bound)
   const double  b,               // distribution parameter b (upper bound)
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileUniform

For the specified *probability*, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileUniform(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileUniform(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns false. Analog of the qcauschy() in R.

```
double  MathQuantileUniform(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse uniform distribution function with the a and b parameters. In case of error it returns false.

```
bool  MathQuantileUniform(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomUniform

Generates a pseudorandom variable distributed according to the law of uniform distribution with the a and b parameters. In case of error it returns NaN.

```
double  MathRandomUniform(
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of uniform distribution with the a and b parameters. In case of error it returns false. Analog of the runif() in R.

```
bool  MathRandomUniform(
   const double  a,              // distribution parameter a (lower bound)
   const double  b,              // distribution parameter b (upper bound)
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsUniform

Calculates the theoretical numerical values of the first 4 moments of the uniform distribution with the a and b parameters.

```
double  MathMomentsUniform(
   const double  a,               // distribution parameter a (lower bound)
   const double  b,               // distribution parameter b (upper bound)
   double&       mean,            // variable for the mean
   double&       variance,        // variable for the variance
   double&       skewness,        // variable for the skewness
   double&       kurtosis,        // variable for the kurtosis
   int&          error_code       // variable for the error code
   );
```

Parameters

a

[in] Distribution parameter a (lower bound).

b

[in] Distribution parameter b (upper bound).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

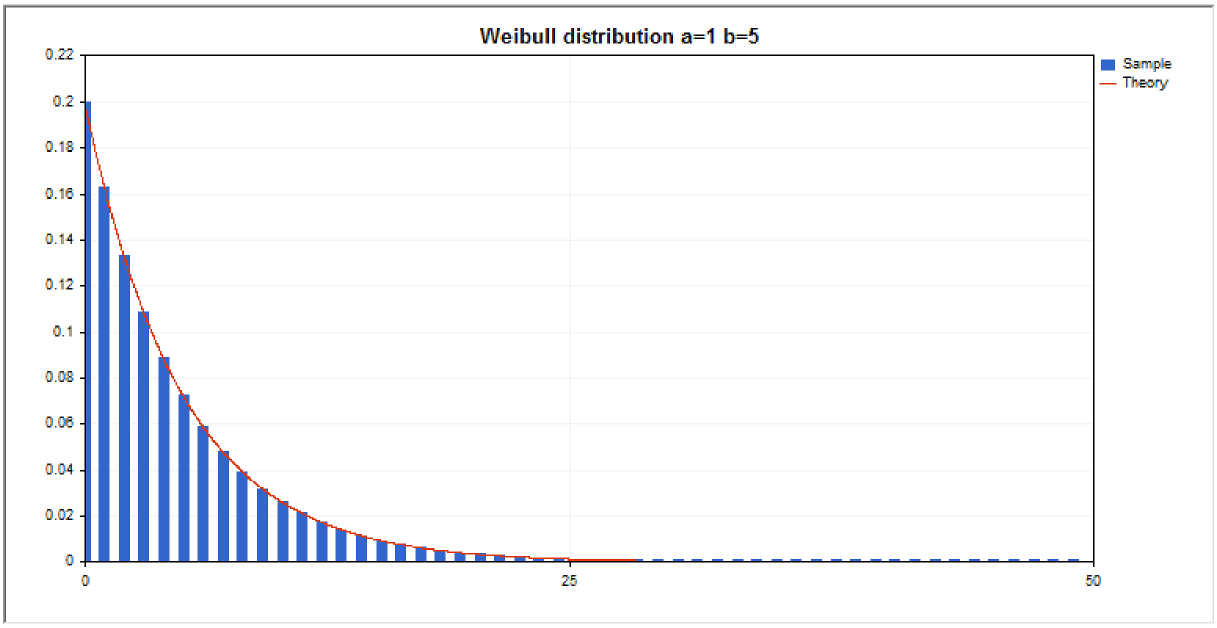
Weibull distribution

This section contains functions for working with Weibull distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Weibull law. The Weibull distribution is defined by the following formula:

$$f_{Weibull}\left(x\middle|a,b\right)=\frac{a}{b}\left(\frac{x}{b}\right)^{a-1}e^{-\left(\frac{x}{b}\right)^{a}}$$

where:

- x — value of the random variable
- a — parameter of the distribution (shape)
- b — parameter of the distribution (scale)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityWeibull	Calculates the probability density function of the Weibull distribution
MathCumulativeDistributionWeibull	Calculates the value of the Weibull probability distribution function
MathQuantileWeibull	Calculates the value of the inverse Weibull distribution function for the specified probability
MathRandomWeibull	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Weibull law
MathMomentsWeibull	Calculates the theoretical numerical values of the first 4 moments of the Weibull distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Weibull.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=1;      // parameter of the distribution (shape)
input double b_par=5;      // parameter of the distribution (scale)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=51;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the Weibull distribution
   MathRandomWeibull(a_par,b_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   step=MathMin(step,(max-min)/ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(min,max,step,x2);
   MathProbabilityDensityWeibull(x2,a_par,b_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
```

```
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Weibull distribution a=%G b=%G",a_par,b_par));
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(max);
   graphic.XAxis().Min(min);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityWeibull

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityWeibull(
   const double  x,             // value of random variable
   const double  a,             // parameter of the distribution (shape)
   const double  b,             // parameter of the distribution (scale)
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityWeibull(
   const double  x,             // value of random variable
   const double  a,             // parameter of the distribution (shape)
   const double  b,             // parameter of the distribution (scale)
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the dweibull() in R.

```
bool  MathProbabilityDensityWeibull(
   const double& x[],            // array with the values of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability density function of Weibull distribution with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityWeibull(
   const double& x[],            // array with the values of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionWeibull

Calculates the value of the Weibull distribution function with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionWeibull(
   const double  x,              // value of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Weibull distribution function with the a and b parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionWeibull(
   const double  x,              // value of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Weibull distribution function with the a and b parameters for an array of random variables x[]. In case of error it returns false. Analog of the pweibull() in R.

```
bool  MathCumulativeDistributionWeibull(
   const double& x[],             // array with the values of random variable
   const double  a,               // parameter of the distribution (shape)
   const double  b,               // parameter of the distribution (scale)
   const bool    tail,            // flag of calculation, if true, then the probabilit
   const bool    log_mode,        // flag to calculate the logarithm of the value, if
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the Weibull distribution function with the a and b parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionWeibull(
   const double& x[],             // array with the values of random variable
   const double  a,               // parameter of the distribution (shape)
   const double  b,               // parameter of the distribution (scale)
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileWeibull

For the specified *probability*, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileWeibull(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns NaN.

```
double  MathQuantileWeibull(
   const double  probability,    // probability value of random variable occurrence
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns false. Analog of the qweibull() in R.

```
double  MathQuantileWeibull(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the value of inverse Weibull distribution function with the a and b parameters. In case of error it returns false.

```
bool  MathQuantileWeibull(
   const double& probability[],  // array with probability values of random variable
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomWeibull

Generates a pseudorandom variable distributed according to the law of Weibull distribution with the a and b parameters. In case of error it returns NaN.

```
double  MathRandomWeibull(
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of Weibull distribution with the a and b parameters. In case of error it returns false. Analog of the rweibull() in R.

```
bool  MathRandomWeibull(
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsWeibull

Calculates the theoretical numerical values of the first 4 moments of the Weibull distribution with the a and b parameters.

```
double  MathMomentsWeibull(
   const double  a,              // parameter of the distribution (shape)
   const double  b,              // parameter of the distribution (scale)
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

a

[in] Parameter of the distribution (scale).

b

[in] Parameter of the distribution (shape).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

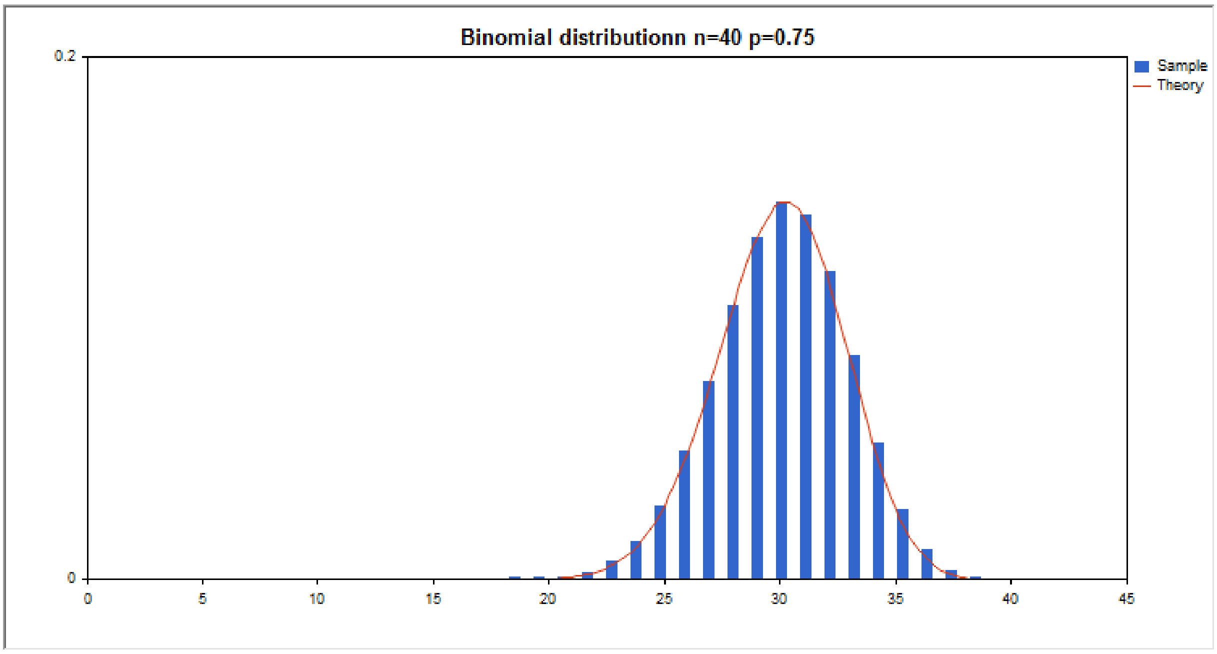
Binomial distribution

This section contains functions for working with binomial distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the binomial law. The binomial distribution is defined by the following formula:

$$f_{Binomial}(x \mid n, p) = \binom{n}{x} p^{x} (1-p)^{n-x}$$

where:

- x — value of the random variable
- n — number of tests
- p — probability of success for each test



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityBinomial	Calculates the probability density function of the binomial distribution
MathCumulativeDistributionBinomial	Calculates the value of the binomial probability distribution function
MathQuantileBinomial	Calculates the value of the inverse binomial distribution function for the specified probability
MathRandomBinomial	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the binomial law
MathMomentsBinomial	Calculates the theoretical numerical values of the first 4 moments of the binomial distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Binomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;        // the number of tests
input double p_par=0.75;      // probability of success for each test
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=20;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the binomial distribution
   MathRandomBinomial(n_par,p_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityBinomial(x2,n_par,p_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Binomial distributionn n=%G p=%G",n_par,p_par)
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

MathProbabilityDensityBinomial

Calculates the value of the probability mass function of binomial distribution with the n and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityBinomial(
   const double  x,            // value of random variable
   const double  n,            // parameter of the distribution (number of tests)
   const double  p,            // parameter of the distribution (probability of event
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability mass function of binomial distribution with the n and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityBinomial(
   const double  x,            // value of random variable
   const double  n,            // parameter of the distribution (number of tests)
   const double  p,            // parameter of the distribution (probability of event
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability mass function of binomial distribution with the n and p parameters for an array of random variables x[]. In case of error it returns false. Analog of the dbinom() in R.

```
bool  MathProbabilityDensityBinomial(
   const double& x[],            // array with the values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of eve
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability mass function of binomial distribution with the n and p parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityBinomial(
   const double& x[],            // array with the values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of eve
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionBinomial

Calculates the value of the probability distribution function for binomial law with the n and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionBinomial(
   const double  x,             // value of random variable
   const double  n,             // parameter of the distribution (number of tests)
   const double  p,             // parameter of the distribution (probability of event
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // flag to calculate the logarithm of the value, if lo
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionBinomial(
   const double  x,             // value of random variable
   const double  n,             // parameter of the distribution (number of tests)
   const double  p,             // parameter of the distribution (probability of event
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for an array of random variables x[]. In case of error it returns false. Analog of the pweibull() in R.

```
bool  MathCumulativeDistributionBinomial(
   const double& x[],            // array with the values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability function
   );
```

Calculates the value of the probability distribution function for binomial law with the n and p parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionBinomial(
   const double& x[],            // array with the values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   double&       result[]        // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation. If true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileBinomial

For the specified *probability*, the function calculates the inverse value of distribution function for binomial law with the n and p parameters. In case of error it returns NaN.

```
double  MathQuantileBinomial(
   const double  probability,    // probability value of random variable occurrence
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the inverse value of distribution function for binomial law with the n and p parameters. In case of error it returns NaN.

```
double  MathQuantileBinomial(
   const double  probability,    // probability value of random variable occurrence
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for binomial law with the n and p parameters. In case of error it returns false. Analog of the qbinom() in R.

```
double  MathQuantileBinomial(
   const double& probability[],  // array with probability values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for binomial law with the n and p parameters. In case of error it returns false.

```
bool  MathQuantileBinomial(
   const double& probability[],  // array with probability values of random variable
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomBinomial

Generates a pseudorandom variable distributed according to the law of binomial distribution with the n and p parameters. In case of error it returns NaN.

```
double  MathRandomBinomial(
   const double  n,             // parameter of the distribution (number of tests)
   const double  p,             // parameter of the distribution (probability of ever
   int&          error_code     // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of binomial distribution with the n and p parameters. In case of error it returns false. Analog of the rweibull() in R.

```
bool  MathRandomBinomial(
   const double  n,             // parameter of the distribution (number of tests)
   const double  p,             // parameter of the distribution (probability of ever
   const int     data_count,    // amount of required data
   double&       result[]       // array with values of pseudorandom variables
   );
```

Parameters

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsBinomial

Calculates the theoretical numerical values of the first 4 moments of the binomial distribution with the n and p parameters.

```
double  MathMomentsBinomial(
   const double  n,              // parameter of the distribution (number of tests)
   const double  p,              // parameter of the distribution (probability of ever
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

n

[in] Parameter of the distribution (number of tests).

p

[in] Parameter of the distribution (probability of event occurrence in one test).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

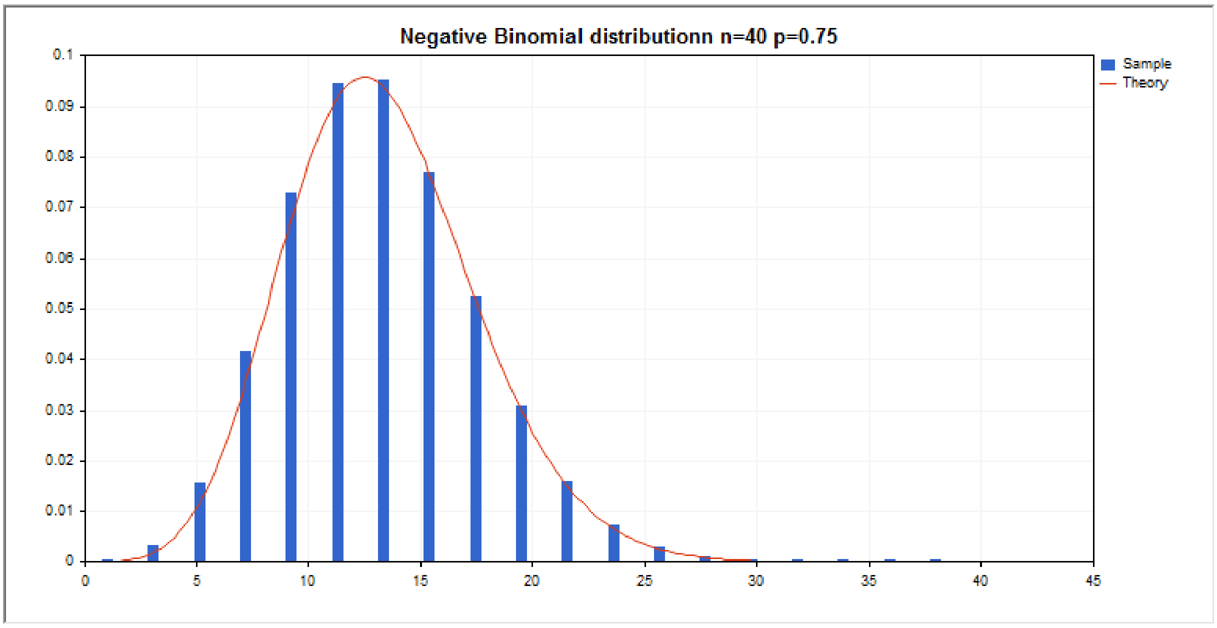
Negative binomial distribution

This section contains functions for working with negative binomial distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the negative binomial law. The negative binomial distribution is defined by the following formula:

$$f_{NegativeBinomial}(x \mid r, p) = \frac{\Gamma(r+x)}{\Gamma(r)\Gamma(x+1)} p^{r} (1-p)^{x}$$

where:

- x — value of the random variable
- r — number of successful tests
- p — probability of success



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityNegativeBinomial	Calculates the probability density function of the negative binomial distribution
MathCumulativeDistributionNegativeBinomial	Calculates the value of the negative binomial probability distribution function
MathQuantileNegativeBinomial	Calculates the value of the inverse negative binomial distribution function for the specified probability
MathRandomNegativeBinomial	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the negative binomial law
MathMomentsNegativeBinomial	Calculates the theoretical numerical values of the first 4 moments of the negative binomial distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NegativeBinomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;        // the number of tests
input double p_par=0.75;      // probability of success for each test
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=19;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the negative binomial distribution
   MathRandomNegativeBinomial(n_par,p_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityNegativeBinomial(x2,n_par,p_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Negative Binomial distributionn n=%G p=%G",n_
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
```

MathProbabilityDensityNegativeBinomial

Calculates the value of the probability mass function of negative binomial distribution with the r and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNegativeBinomial(
   const double  x,             // value of random variable (integer)
   const double  r,             // number of successful tests
   const double  p,             // probability of success
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability mass function of negative binomial distribution with the r and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityNegativeBinomial(
   const double  x,             // value of random variable (integer)
   const double  r,             // number of successful tests
   const double  p,             // probability of success
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability mass function of negative binomial distribution with the r and p parameters for an array of random variables x[]. In case of error it returns false. Analog of the dnbinom() in R.

```
bool  MathProbabilityDensityNegativeBinomial(
   const double& x[],            // array with the values of random variable
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability density funct
   );
```

Calculates the value of the probability mass function of negative binomial distribution with the r and p parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityNegativeBinomial(
   const double& x[],            // array with the values of random variable
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   double&       result[]        // array for values of the probability density funct
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

r

[in] Number of successful tests

p

[in] Probability of success.

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionNegativeBinomial

Calculates the value of the probability distribution function for negative binomial law with the r and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNegativeBinomial(
   const double  x,             // value of random variable (integer)
   const double  r,             // number of successful tests
   const double  p,             // probability of success
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // flag to calculate the logarithm of the value, if lo
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for negative binomial law with the r and p parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionNegativeBinomial(
   const double  x,             // value of random variable (integer)
   const double  r,             // number of successful tests
   const double  p,             // probability of success
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for negative binomial law with the r and p parameters for an array of random variables x[]. In case of error it returns false. Analog of the pweibull() in R.

```
bool  MathCumulativeDistributionNegativeBinomial(
   const double& x[],             // array with the values of random variable
   const double  r,               // number of successful tests
   const double  p,               // probability of success
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the probability distribution function for negative binomial law with the r and p parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionNegativeBinomial(
   const double& x[],             // array with the values of random variable
   const double  r,               // number of successful tests
   const double  p,               // probability of success
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

r

[in] Number of successful tests.

p

[in] Probability of success.

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileNegativeBinomial

For the specified *probability*, the function calculates the inverse value of distribution function for negative binomial law with the r and p parameters. In case of error it returns NaN.

```
double  MathQuantileNegativeBinomial(
   const double  probability,    // probability value of random variable occurrence
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the inverse value of distribution function for negative binomial law with the r and p parameters. In case of error it returns NaN.

```
double  MathQuantileNegativeBinomial(
   const double  probability,    // probability value of random variable occurrence
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for negative binomial law with the r and p parameters. In case of error it returns false. Analog of the qnbinom() in R.

```
double  MathQuantileNegativeBinomial(
   const double& probability[],  // array with probability values of random variable
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for negative binomial law with the r and p parameters. In case of error it returns false.

```
bool  MathQuantileNegativeBinomial(
   const double& probability[],  // array with probability values of random variable
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

r

[in] Number of successful tests.

p

[in] Probability of success.

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomNegativeBinomial

Generates a pseudorandom variable distributed according to the law of negative binomial distribution with the r and p parameters. In case of error it returns NaN.

```
double  MathRandomNegativeBinomial(
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   int&          error_code      // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of negative binomial distribution with the r and p parameters. In case of error it returns false. Analog of the rweibull() in R.

```
bool  MathRandomNegativeBinomial(
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   const int     data_count,     // amount of required data
   double&       result[]        // array with values of pseudorandom variables
   );
```

Parameters

r

[in] Number of successful tests.

p

[in] Probability of success.

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsNegativeBinomial

Calculates the theoretical numerical values of the first 4 moments of the negative binomial distribution with the r and p parameters.

```
double  MathMomentsNegativeBinomial(
   const double  r,              // number of successful tests
   const double  p,              // probability of success
   double&       mean,           // variable for the mean
   double&       variance,       // variable for the variance
   double&       skewness,       // variable for the skewness
   double&       kurtosis,       // variable for the kurtosis
   int&          error_code      // variable for the error code
   );
```

Parameters

r

[in] Number of successful tests.

p

[in] Probability of success.

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

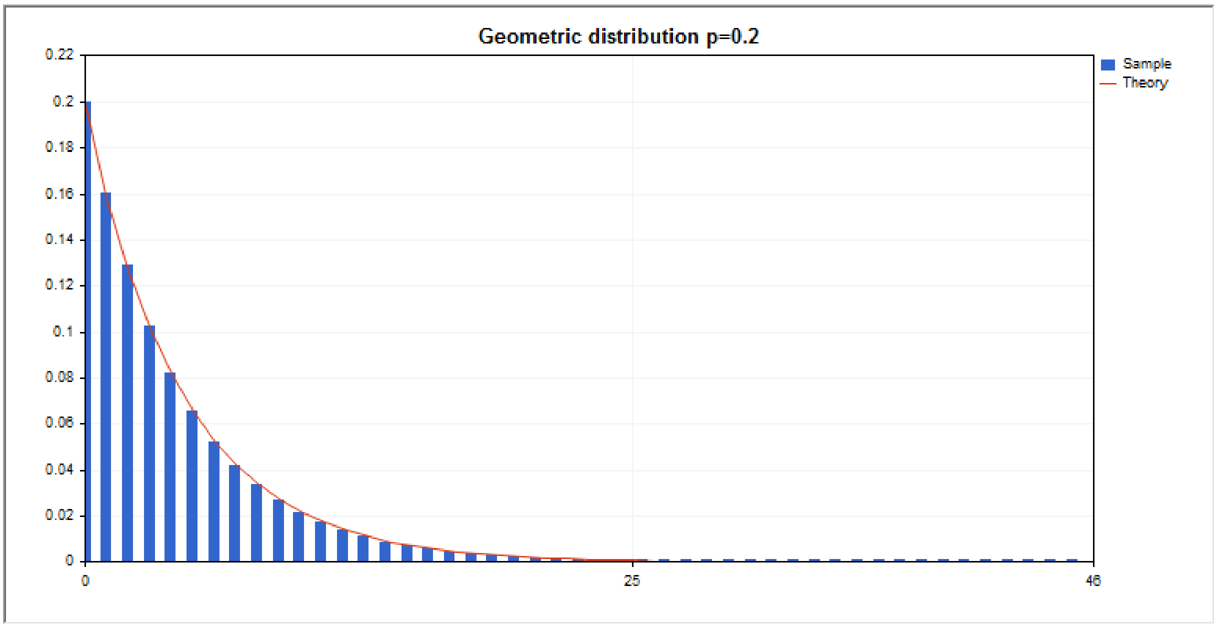
Geometric distribution

This section contains functions for working with geometric distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the geometric law. The geometric distribution is defined by the following formula:

$$f_{Geometric}(x \mid p) = p(1-p)^{x}$$

where:

- x — value of the random variable (integer)
- p — probability of event occurrence in one test



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityGeometric	Calculates the probability density function of the geometric distribution
MathCumulativeDistributionGeometric	Calculates the value of the geometric probability distribution function
MathQuantileGeometric	Calculates the value of the inverse geometric distribution function for the specified probability
MathRandomGeometric	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the geometric law
MathMomentsGeometric	Calculates the theoretical numerical values of the first 4 moments of the geometric distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Geometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double p_par=0.2;      // probability of event occurrence in one test
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=47;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the geometric distribution
   MathRandomGeometric(p_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(0,ncells,1,x2);
   MathProbabilityDensityGeometric(x2,p_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Geometric distribution p=%G",p_par));
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the X axis
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Max(max);
   graphic.XAxis().Min(min);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+i*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
```

```
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityGeometric

Calculates the value of the probability mass function of geometric distribution with the p parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityGeometric(
   const double  x,             // value of random variable (integer)
   const double  p,             // parameter of the distribution (probability of event
   const bool    log_mode,      // calculate the logarithm of the value, if log_mode=t
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityGeometric(
   const double  x,             // value of random variable (integer)
   const double  p,             // parameter of the distribution (probability of event
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for an array of random variables x[]. In case of error it returns false. Analog of the dgeom() in R.

```
bool  MathProbabilityDensityGeometric(
   const double& x[],            // array with the values of random variable
   const double  p,              // parameter of the distribution (probability of ever
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   double&       result[]        // array for values of the probability density functi
   );
```

Calculates the value of the probability mass function of geometric distribution with the p parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityGeometric(
   const double& x[],            // array with the values of random variable
   const double  p,              // parameter of the distribution (probability of ever
   double&       result[]        // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionGeometric

Calculates the value of the probability distribution function for geometric law with the p parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionGeometric(
   const double  x,              // value of random variable (integer)
   const double  p,              // parameter of the distribution (probability of event
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if lo
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function for geometric law with the p parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionGeometric(
   const double  x,              // value of random variable (integer)
   const double  p,              // parameter of the distribution (probability of event
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability distribution function for geometric law with the p parameter for an array of random variables x[]. In case of error it returns false. Analog of the pgeom() in R.

```
bool  MathCumulativeDistributionGeometric(
   const double& x[],             // array with the values of random variable
   const double  p,               // parameter of the distribution (probability of even
   const bool    tail,            // flag of calculation, if true, then the probability
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability function
   );
```

Calculates the value of the probability distribution function for geometric law with the p parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionGeometric(
   const double& x[],             // array with the values of random variable
   const double  p,               // parameter of the distribution (probability of even
   double&       result[]         // array for values of the probability function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if tail=true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability function.

MathQuantileGeometric

For the specified *probability*, the function calculates the inverse value of distribution function for geometric law with the p parameter. In case of error it returns NaN.

```
double  MathQuantileGeometric(
   const double  probability,    // probability value of random variable occurrence
   const double  p,              // parameter of the distribution (probability of ever
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the inverse value of distribution function for geometric law with the p parameter. In case of error it returns NaN.

```
double  MathQuantileGeometric(
   const double  probability,    // probability value of random variable occurrence
   const double  p,              // parameter of the distribution (probability of ever
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for geometric law with the p parameter. In case of error it returns false. Analog of the qgeom() in R.

```
double  MathQuantileGeometric(
   const double& probability[],  // array with probability values of random variable
   const double  p,              // parameter of the distribution (probability of ever
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for geometric law with the p parameter. In case of error it returns false.

```
bool  MathQuantileGeometric(
   const double& probability[],  // array with probability values of random variable
   const double  p,              // parameter of the distribution (probability of ever
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

p

[in] Parameter of the distribution (probability of event occurrence in one test).

tail

[in] Flag of calculation, if false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomGeometric

Generates a pseudorandom variable distributed according to the law of geometric distribution with the p parameter. In case of error it returns NaN.

```
double  MathRandomGeometric(
   const double  p,                // parameter of the distribution (probability of ever
   int&          error_code        // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of geometric distribution with the p parameter. In case of error it returns false. Analog of the rgeom() in R.

```
bool  MathRandomGeometric(
   const double  p,                // parameter of the distribution (probability of ever
   const int     data_count,       // amount of required data
   double&       result[]          // array with values of pseudorandom variables
   );
```

Parameters

p

[in] Parameter of the distribution (probability of event occurrence in one test).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsGeometric

Calculates the theoretical numerical values of the first 4 moments of the geometric distribution with the p parameter.

```
double  MathMomentsGeometric(
   const double  p,            // parameter of the distribution (probability of ever
   double&       mean,         // variable for the mean
   double&       variance,     // variable for the variance
   double&       skewness,     // variable for the skewness
   double&       kurtosis,     // variable for the kurtosis
   int&          error_code    // variable for the error code
   );
```

Parameters

p

[in] Parameter of the distribution (probability of event occurrence in one test).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

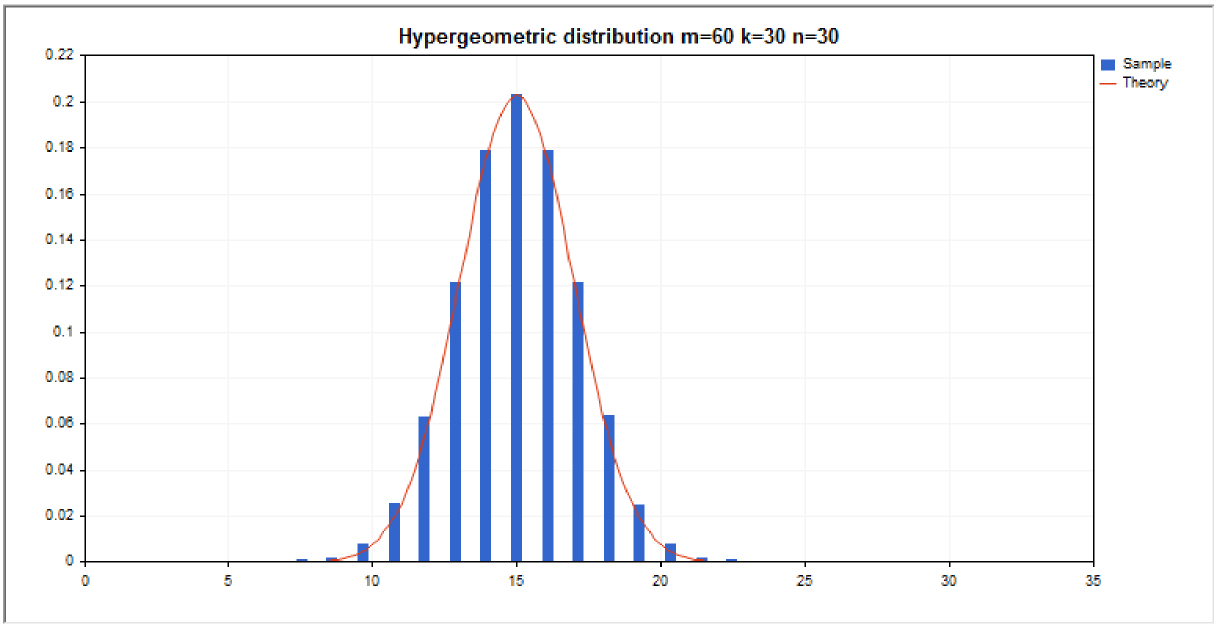
Hypergeometric distribution

This section contains functions for working with hypergeometric distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the hypergeometric law. The hypergeometric distribution is defined by the following formula:

$$f_{Hypergeometric}(x\,|\,m,k,n)=\frac{\binom{k}{x}\binom{m-k}{n-x}}{\binom{m}{n}}$$

where:

- x — value of the random variable (integer)
- m — total number of objects
- k — number of objects with the desired characteristic
- n — number of object draws



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityHypergeometric	Calculates the probability density function of the hypergeometric distribution
MathCumulativeDistributionHypergeometric	Calculates the value of the hypergeometric probability distribution function
MathQuantileHypergeometric	Calculates the value of the inverse hypergeometric distribution function for the specified probability
MathRandomHypergeometric	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the hypergeometric law

Function	Description
MathMomentsHypergeometric	Calculates the theoretical numerical values of the first 4 moments of the hypergeometric distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Hypergeometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double m_par=60;      // the total number of objects
input double k_par=30;      // the number of objects with the desired characteristic
input double n_par=30;      // the number of object draws
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=1000000;       // the number of values in the sample
   int ncells=15;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the hypergeometric distribution
   MathRandomHypergeometric(m_par,k_par,n_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(0,n_par,1,x2);
   MathProbabilityDensityHypergeometric(x2,m_par,k_par,n_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Hypergeometric distribution m=%G k=%G n=%G",m_
   graphic.BackgroundMainSize(16);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
  {
//--- calculate the absolute range of the sequence to obtain the precision of normaliz
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityHypergeometric

Calculates the value of the probability mass function of hypergeometric distribution with the m, k and n parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityHypergeometric(
   const double  x,            // value of random variable (integer)
   const double  m,            // total number of objects (integer)
   const double  k,            // number of objects with the desired characteristic
   const double  n,            // number of object draws (integer)
   const bool    log_mode,     // calculate the logarithm of the value, if log_mode=t
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability mass function of hypergeometric distribution with the m, k and n parameters for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityHypergeometric(
   const double  x,            // value of random variable (integer)
   const double  m,            // total number of objects (integer)
   const double  k,            // number of objects with the desired characteristic
   const double  n,            // number of object draws (integer)
   int&          error_code    // variable to store the error code
   );
```

Calculates the value of the probability mass function of hypergeometric distribution with the m, k and n parameters for an array of random variables x[]. In case of error it returns false. Analog of the dhyper() in R.

```
bool  MathProbabilityDensityHypergeometric(
   const double& x[],           // array with the values of random variable
   const double  m,             // total number of objects (integer)
   const double  k,             // number of objects with the desired characteristic
   const double  n,             // number of object draws (integer)
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   double&       result[]       // array for values of the probability density functi
   );
```

Calculates the value of the probability mass function of hypergeometric distribution with the m, k and n parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityHypergeometric(
   const double& x[],           // array with the values of random variable
   const double  m,             // total number of objects (integer)
   const double  k,             // number of objects with the desired characteristic
   const double  n,             // number of object draws (integer)
   double&       result[]       // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionHypergeometric

Calculates the value of the probability distribution function for hypergeometric law with the m, k and n parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionHypergeometric(
   const double  x,             // value of random variable (integer)
   const double  m,             // total number of objects (integer)
   const double  k,             // number of objects with the desired characteristic
   const double  n,             // number of object draws (integer)
   const bool    tail,          // flag of calculation, if true, then the probability
   const bool    log_mode,      // flag to calculate the logarithm of the value, if l
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for hypergeometric law with the m, k and n parameters for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionHypergeometric(
   const double  x,             // value of random variable (integer)
   const double  m,             // total number of objects (integer)
   const double  k,             // number of objects with the desired characteristic
   const double  n,             // number of object draws (integer)
   int&          error_code     // variable to store the error code
   );
```

Calculates the value of the probability distribution function for hypergeometric law with the m, k and n parameters for an array of random variables x[]. In case of error it returns false. Analog of the dhyper() in R.

```
bool  MathCumulativeDistributionHypergeometric(
   const double& x[],            // array with the values of random variable
   const double  m,              // total number of objects (integer)
   const double  k,              // number of objects with the desired characteristic
   const double  n,              // number of object draws (integer)
   const bool    tail,           // flag of calculation, if true, then the probabilit
   const bool    log_mode,       // flag to calculate the logarithm of the value, if
   double&       result[]        // array for values of the distribution function
   );
```

Calculates the value of the probability distribution function for hypergeometric law with the m, k and n parameters for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionHypergeometric(
   const double& x[],            // array with the values of random variable
   const double  m,              // total number of objects (integer)
   const double  k,              // number of objects with the desired characteristic
   const double  n,              // number of object draws (integer)
   double&       result[]        // array for values of the distribution function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the distribution function.

MathQuantileHypergeometric

For the specified *probability*, the function calculates the inverse value of distribution function for hypergeometric law with the m, k and n parameters. In case of error it returns NaN.

```
double  MathQuantileHypergeometric(
   const double  probability,     // probability value of random variable occurrence
   const double  m,               // total number of objects (integer)
   const double  k,               // number of objects with the desired characteristic
   const double  n,               // number of object draws (integer)
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability*, the function calculates the inverse value of distribution function for hypergeometric law with the m, k and n parameters. In case of error it returns NaN.

```
double  MathQuantileHypergeometric(
   const double  probability,     // probability value of random variable occurrence
   const double  m,               // total number of objects (integer)
   const double  k,               // number of objects with the desired characteristic
   const double  n,               // number of object draws (integer)
   int&          error_code       // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for hypergeometric law with the m, k and n parameters. In case of error it returns false. Analog of the qhyper() in R.

```
double  MathQuantileHypergeometric(
   const double& probability[],   // array with probability values of random variable
   const double  m,               // total number of objects (integer)
   const double  k,               // number of objects with the desired characteristic
   const double  n,               // number of object draws (integer)
   const bool    tail,            // flag of calculation, if false, then calculation is
   const bool    log_mode,        // flag of calculation, if log_mode=true, calculation
   double&       result[]         // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of distribution function for hypergeometric law with the m, k and n parameters. In case of error it returns false.

```
bool  MathQuantileHypergeometric(
   const double& probability[],   // array with probability values of random variable
   const double  m,               // total number of objects (integer)
   const double  k,               // number of objects with the desired characteristic
   const double  n,               // number of object draws (integer)
   double&       result[]         // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

tail

[in] Flag of calculation, if tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomHypergeometric

Generates a pseudorandom variable distributed according to the law of hypergeometric distribution with the m, n and k parameters. In case of error it returns NaN.

```
double  MathRandomHypergeometric(
   const double  m,                // total number of objects (integer)
   const double  k,                // number of objects with the desired characteristic
   const double  n,                // number of object draws (integer)
   int&          error_code        // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of hypergeometric distribution with the m, n and k parameters. In case of error it returns false. Analog of the rgeom() in R.

```
bool  MathRandomHypergeometric(
   const double  m,                // total number of objects (integer)
   const double  k,                // number of objects with the desired characteristic
   const double  n,                // number of object draws (integer)
   const int     data_count,       // amount of required data
   double&       result[]          // array with values of pseudorandom variables
   );
```

Parameters

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsHypergeometric

Calculates the theoretical numerical values of the first 4 moments of the hypergeometric distribution with the m, n and k parameters.

```
double  MathMomentsHypergeometric(
   const double  m,                // total number of objects (integer)
   const double  k,                // number of objects with the desired characteristic
   const double  n,                // number of object draws (integer)
   double&       mean,             // variable for the mean
   double&       variance,         // variable for the variance
   double&       skewness,         // variable for the skewness
   double&       kurtosis,         // variable for the kurtosis
   int&          error_code        // variable for the error code
   );
```

Parameters

m

[in] Total number of objects (integer).

k

[in] Number of objects with the desired characteristic (integer).

n

[in] Number of object draws (integer).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

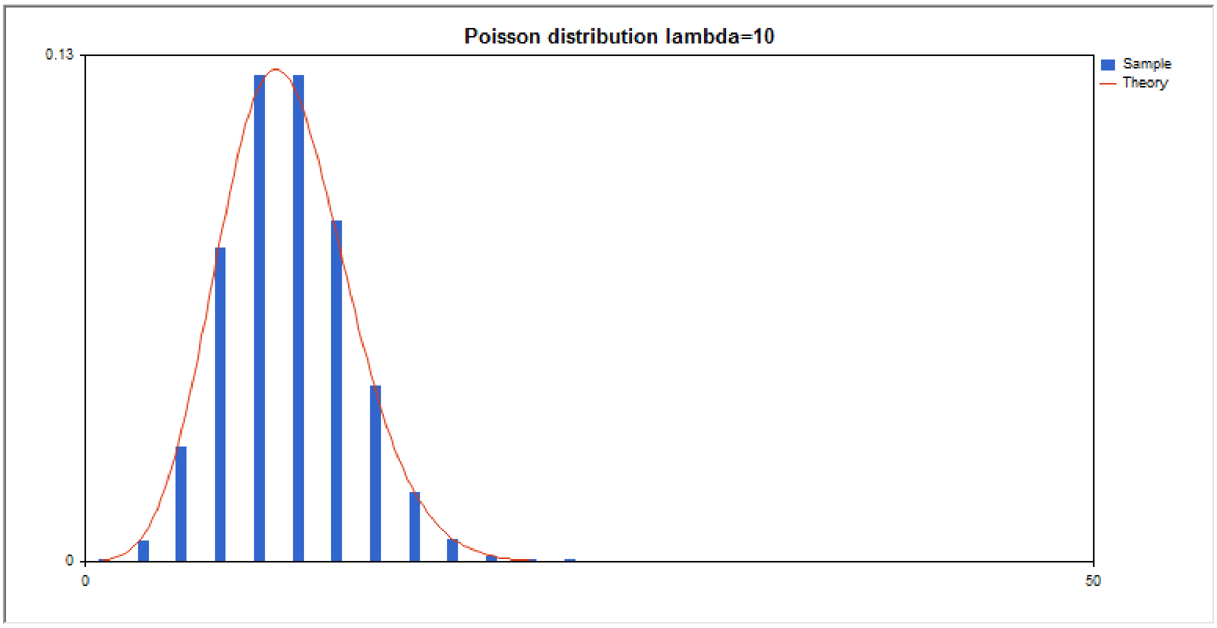
Poisson distribution

This section contains functions for working with Poisson distribution. They allow to calculate density, probability, quantiles and to generate pseudo-random numbers distributed according to the Poisson law. The Poisson distribution is defined by the following formula:

$$f_{Poisson}(x|\lambda)=\frac{\lambda^x}{x!}e^{-\lambda}$$

where:

- x — value of the random variable
- λ — parameter of the distribution (mean)



In addition to the calculation of the individual random variables, the library also implements the ability to work with arrays of random variables.

Function	Description
MathProbabilityDensityPoisson	Calculates the probability density function of the Poisson distribution
MathCumulativeDistributionPoisson	Calculates the value of the Poisson probability distribution function
MathQuantilePoisson	Calculates the value of the inverse Poisson distribution function for the specified probability
MathRandomPoisson	Generates a pseudorandom variable/array of pseudorandom variables distributed according to the Poisson law
MathMomentsPoisson	Calculates the theoretical numerical values of the first 4 moments of the Poisson distribution

Example:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Poisson.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double lambda_par=10;      // parameter of the distribution (mean)
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- hide the price chart
   ChartSetInteger(0,CHART_SHOW,false);
//--- initialize the random number generator
   MathSrand(GetTickCount());
//--- generate a sample of the random variable
   long chart=0;
   string name="GraphicNormal";
   int n=100000;       // the number of values in the sample
   int ncells=13;       // the number of intervals in the histogram
   double x[];          // centers of the histogram intervals
   double y[];          // the number of values from the sample falling within the int
   double data[];       // sample of random values
   double max,min;      // the maximum and minimum values in the sample
//--- obtain a sample from the Poisson distribution
   MathRandomPoisson(lambda_par,n,data);
//--- calculate the data to plot the histogram
   CalculateHistogramArray(data,x,y,max,min,ncells);
//--- obtain the sequence boundaries and the step for plotting the theoretical curve
   double step;
   GetMaxMinStepValues(max,min,step);
   PrintFormat("max=%G min=%G",max,min);
//--- obtain the theoretically calculated data at the interval of [min,max]
   double x2[];
   double y2[];
   MathSequence(0,int(MathCeil(max)),1,x2);
   MathProbabilityDensityPoisson(x2,lambda_par,false,y2);
//--- set the scale
   double theor_max=y2[ArrayMaximum(y2)];
   double sample_max=y[ArrayMaximum(y)];
   double k=sample_max/theor_max;
   for(int i=0; i<ncells; i++)
      y[i]/=k;
//--- output charts
   CGraphic graphic;
   if(ObjectFind(chart,name)<0)
      graphic.Create(chart,name,0,0,0,780,380);
   else
      graphic.Attach(chart,name);
   graphic.BackgroundMain(StringFormat("Poisson distribution lambda=%G",lambda_par));
   graphic.BackgroundMainSize(16);
//--- disable automatic scaling of the Y axis
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Max(NormalizeDouble(theor_max,2));
   graphic.YAxis().Min(0);
//--- plot all curves
   graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- and now plot the theoretical curve of the distribution density
   graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
   graphic.CurvePlotAll();
//--- plot all curves
   graphic.Update();
  }
//+------------------------------------------------------------------+
//|  Calculate frequencies for data set                              |
//+------------------------------------------------------------------+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequenc
                             double &maxv,double &minv,const int cells=10)
  {
   if(cells<=1) return (false);
   int size=ArraySize(data);
   if(size<cells*10) return (false);
   minv=data[ArrayMinimum(data)];
   maxv=data[ArrayMaximum(data)];
   double range=maxv-minv;
   double width=range/cells;
   if(width==0) return false;
   ArrayResize(intervals,cells);
   ArrayResize(frequency,cells);
//--- define the interval centers
   for(int i=0; i<cells; i++)
     {
      intervals[i]=minv+(i+0.5)*width;
      frequency[i]=0;
     }
//--- fill the frequencies of falling within the interval
   for(int i=0; i<size; i++)
     {
      int ind=int((data[i]-minv)/width);
      if(ind>=cells) ind=cells-1;
      frequency[ind]++;
     }
   return (true);
  }
//+------------------------------------------------------------------+
//|  Calculates values for sequence generation                       |
//+------------------------------------------------------------------+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
```

```
  {
//--- calculate the absolute range of the sequence to obtain the precision of normali
   double range=MathAbs(maxv-minv);
   int degree=(int)MathRound(MathLog10(range));
//--- normalize the maximum and minimum values to the specified precision
   maxv=NormalizeDouble(maxv,degree);
   minv=NormalizeDouble(minv,degree);
//--- sequence generation step is also set based on the specified precision
   stepv=NormalizeDouble(MathPow(10,-degree),degree);
   if((maxv-minv)/stepv<10)
      stepv/=10.;
  }
```

MathProbabilityDensityPoisson

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityPoisson(
   const double  x,              // value of random variable (integer)
   const double  lambda,         // parameter of the distribution (mean)
   const bool    log_mode,       // calculate the logarithm of the value, if log_mode=t
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for a random variable x. In case of error it returns NaN.

```
double  MathProbabilityDensityPoisson(
   const double  x,              // value of random variable (integer)
   const double  lambda,         // parameter of the distribution (mean)
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for an array of random variables x[]. In case of error it returns false. Analog of the dhyper() in R.

```
bool  MathProbabilityDensityPoisson(
   const double& x[],             // array with the values of random variable
   const double  lambda,          // parameter of the distribution (mean)
   const bool    log_mode,        // flag to calculate the logarithm of the value, if l
   double&       result[]         // array for values of the probability density functi
   );
```

Calculates the value of the probability mass function of Poisson distribution with the lambda parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathProbabilityDensityPoisson(
   const double& x[],             // array with the values of random variable
   const double  lambda,          // parameter of the distribution (mean)
   double&       result[]         // array for values of the probability density functi
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

lambda

[in] Parameter of the distribution (mean).

log_mode

[in] Flag to calculate the logarithm of the value. If log_mode=true, then the natural logarithm of the probability density is returned.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the probability density function.

MathCumulativeDistributionPoisson

Calculates the value of the Poisson distribution function with the lambda parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionPoisson(
   const double  x,              // value of random variable (integer)
   const double  lambda,         // parameter of the distribution (mean)
   const bool    tail,           // flag of calculation, if true, then the probability
   const bool    log_mode,       // flag to calculate the logarithm of the value, if l
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Poisson distribution function with the lambda parameter for a random variable x. In case of error it returns NaN.

```
double  MathCumulativeDistributionPoisson(
   const double  x,              // value of random variable (integer)
   const double  lambda,         // parameter of the distribution (mean)
   int&          error_code      // variable to store the error code
   );
```

Calculates the value of the Poisson distribution function with the lambda parameter for an array of random variables x[]. In case of error it returns false. Analog of the dhyper() in R.

```
bool  MathCumulativeDistributionPoisson(
   const double& x[],             // array with the values of random variable
   const double  lambda,          // parameter of the distribution (mean)
   const bool    tail,            // flag of calculation, if true, then the probabilit
   const bool    log_mode,        // flag to calculate the logarithm of the value, if
   double&       result[]         // array for values of the distribution function
   );
```

Calculates the value of the Poisson distribution function with the lambda parameter for an array of random variables x[]. In case of error it returns false.

```
bool  MathCumulativeDistributionPoisson(
   const double& x[],             // array with the values of random variable
   const double  lambda,          // parameter of the distribution (mean)
   double&       result[]         // array for values of the distribution function
   );
```

Parameters

x

[in] Value of random variable.

x[]

[in] Array with the values of random variable.

lambda

[in] Parameter of the distribution (mean).

tail

[in] Flag of calculation, if true, then the probability of random variable not exceeding x is calculated.

log_mode

[in] Flag to calculate the logarithm of the value, if log_mode=true, then the natural logarithm of the probability is calculated.

error_code

[out] Variable to store the error code.

result[]

[out] Array for values of the distribution function.

MathQuantilePoisson

For the specified *probability*, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns NaN.

```
double  MathQuantilePoisson(
   const double  probability,    // probability value of random variable occurrence
   const double  lambda,         // parameter of the distribution (mean)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability*, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns NaN.

```
double  MathQuantilePoisson(
   const double  probability,    // probability value of random variable occurrence
   const double  lambda,         // parameter of the distribution (mean)
   int&          error_code      // variable to store the error code
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns false. Analog of the qhyper() in R.

```
double  MathQuantilePoisson(
   const double& probability[],  // array with probability values of random variable
   const double  lambda,         // parameter of the distribution (mean)
   const bool    tail,           // flag of calculation, if false, then calculation is
   const bool    log_mode,       // flag of calculation, if log_mode=true, calculation
   double&       result[]        // array with values of quantiles
   );
```

For the specified *probability[]* array of probability values, the function calculates the inverse value of Poisson distribution function with the lambda parameter. In case of error it returns false.

```
bool  MathQuantilePoisson(
   const double& probability[],  // array with probability values of random variable
   const double  lambda,         // parameter of the distribution (mean)
   double&       result[]        // array with values of quantiles
   );
```

Parameters

probability

[in] Probability value of random variable.

probability[]

[in] Array with probability values of random variable.

lambda

[in] Parameter of the distribution (mean).

tail

[in] Flag of calculation, if tail=false, then calculation is performed for 1.0-probability.

log_mode

[in] Flag of calculation, if log_mode=true, calculation is performed for Exp(probability).

error_code

[out] Variable to get the error code.

result[]

[out] Array with values of quantiles.

MathRandomPoisson

Generates a pseudorandom variable distributed according to the law of Poisson distribution with the lambda parameter. In case of error it returns NaN.

```
double  MathRandomPoisson(
   const double  lambda,          // parameter of the distribution (mean)
   int&          error_code       // variable to store the error code
   );
```

Generates pseudorandom variables distributed according to the law of Poisson distribution with the lambda parameter. In case of error it returns false. Analog of the rgeom() in R.

```
bool  MathRandomPoisson(
   const double  lambda,          // parameter of the distribution (mean)
   const int     data_count,      // amount of required data
   double&       result[]         // array with values of pseudorandom variables
   );
```

Parameters

lambda

[in] Parameter of the distribution (mean).

error_code

[out] Variable to store the error code.

data_count

[out] Amount of required data.

result[]

[out] Array to obtain the values of pseudorandom variables.

MathMomentsPoisson

Calculates the theoretical numerical values of the first 4 moments of the Poisson distribution with the lambda parameter.

```
double  MathMomentsPoisson(
   const double  lambda,          // parameter of the distribution (mean)
   double&       mean,            // variable for the mean
   double&       variance,        // variable for the variance
   double&       skewness,        // variable for the skewness
   double&       kurtosis,        // variable for the kurtosis
   int&          error_code       // variable for the error code
   );
```

Parameters

lambda

[in] Parameter of the distribution (mean).

mean

[out] Variable to get the mean value.

variance

[out] Variable to get the variance.

skewness

[out] Variable to get the skewness.

kurtosis

[out] Variable to get the kurtosis.

error_code

[out] Variable to get the error code.

Return Value

Returns true if calculation of the moments has been successful, otherwise false.

Subfunctions

Group of functions that perform basic mathematical operations: calculation of the gamma function, beta function, factorial, exponential, logarithms with different bases, square root, etc.

They provide the ability to process both individual numeric values (real and integer) and arrays of such values (with output of the results to a separate or the original array).

Function	Description
MathRandomNonZero	Returns a random number with a floating point in the range from 0.0 to 1.0.
MathMoments	Calculates the first 4 moments of array elements: mean, variance, skewness, kurtosis.
MathPowInt	Raises a number to the specified integer power.
MathFactorial	Calculates the factorial of the specified integer.
MathTrunc	Calculates the integer part of the specified number or array elements.
MathRound	Rounds a number or an array of numbers to the specified number of decimal places.
MathArctan2	Calculates the angle, the tangent of which is equal to the ratio of the two specified numbers in the range of [-pi, pi].
MathGamma	Calculates the value of the gamma function.
MathGammaLog	Calculates the logarithm of the gamma function.
MathBeta	Calculates the value of the beta function.
MathBetaLog	Calculates the logarithm of the beta function.
MathBetaIncomplete	Calculates the value of the incomplete beta function.
MathGammaIncomplete	Calculates the value of the incomplete gamma function.
MathBinomialCoefficient	Calculates the binomial coefficient.
MathBinomialCoefficientLog	Calculates the logarithm of the binomial coefficient.
MathHypergeometric2F2	Calculates the value of the hypergeometric function.

Function	Description
MathSequence	Generates a sequence based on values: the first element, the last element, the step of the sequence.
MathSequenceByCount	Generates a sequence based on values: the first element, the last element, the number of elements in the sequence.
MathReplicate	Generates a repeating sequence of values.
MathReverse	Generates an array of values with reverse order of elements.
MathIdentical	Compares two arrays of values and returns true if all elements match.
MathUnique	Generates an array with unique values only.
MathQuickSortAscending	Function for sorting in ascending order.
MathQuickSortDescending	Function for sorting in descending order.
MathQuickSort	Function for sorting.
MathOrder	Generates an array with permutation according to order of the array elements after sorting.
MathBitwiseNot	Calculates the result of bitwise NOT operation for array elements.
MathBitwiseAnd	Calculates the result of bitwise AND operation for elements of arrays.
MathBitwiseOr	Calculates the result of bitwise OR operation for elements of arrays.
MathBitwiseXor	Calculates the result of bitwise XOR operation for elements of arrays.
MathBitwiseShiftL	Calculates the result of bitwise SHL operation for array elements.
MathBitwiseShiftR	Calculates the result of bitwise SHR operation for array elements.
MathCumulativeSum	Generates an array with the cumulative sums.
MathCumulativeProduct	Generates an array with the cumulative products.
MathCumulativeMin	Generates an array with the cumulative minima.
MathCumulativeMax	Generates an array with the cumulative maxima.

Function	Description
MathSin	Calculates the values of the sin(x) function for array elements.
MathCos	Calculates the values of the cos(x) function for array elements.
MathTan	Calculates the values of the tan(x) function for array elements.
MathArcsin	Calculates the values of the arcsin(x) function for array elements.
MathArccos	Calculates the values of the arccos(x) function for array elements.
MathArctan	Calculates the values of the arctan(x) function for array elements.
MathSinPi	Calculates the values of the sin(pi*x) function for array elements.
MathCosPi	Calculates the values of the cos(pi*x) function for array elements.
MathTanPi	Calculates the values of the tan(pi*x) function for array elements.
MathAbs	Calculates the absolute values of array elements.
MathCeil	Returns the nearest larger integers for array elements.
MathFloor	Returns the nearest smaller integers for array elements.
MathSqrt	Calculates the square roots of array elements.
MathExp	Calculates the values of the exp(x) function for array elements.
MathPow	Calculates the values of the pow(x, power) function for array elements.
MathLog	Calculates the values of the log(x) function for array elements.
MathLog2	Calculates the logarithm to the base 2 for the array elements.
MathLog10	Calculates the logarithm to the base 10 for the array elements.
MathDifference	Generates an array with element differences of y[i]=x[i+lag]-x[i].

Function	Description
MathSample	Generates a random sample from the array elements.
MathTukeySummary	Calculates the Tukey's five-number summary for the array elements.
MathRange	Calculates the minima and maxima of array elements.
MathMin	Returns the minimum value of all array elements.
MathMax	Returns the maximum value of all array elements.
MathSum	Returns the sum of array elements.
MathProduct	Returns the product of array elements.
MathStandardDeviation	Calculates the standard deviation of array elements.
MathAverageDeviation	Calculates the average absolute deviation of array elements.
MathMedian	Calculates the median value of array elements.
MathMean	Calculates the mean values of array elements.
MathVariance	Calculates the variance of the array elements.
MathSkewness	Calculates the skewness of the array elements.
MathKurtosis	Calculates the kurtosis of the array elements.
MathLog1p	Calculates the values of the log(1+x) function for array elements.
MathExpm1	Calculates the values of the exp(x)-1 function for array elements.
MathSinh	Calculates the values of the sinh(x) function for array elements.
MathCosh	Calculates the values of the cosh(x) function for array elements.
MathTanh	Calculates the values of the tanh(x) function for array elements.
MathArcsinh	Calculates the values of the arcsinh(x) function for array elements.

Function	Description
MathArccosh	Calculates the values of the arccosh(x) function for array elements.
MathArctanh	Calculates the values of the arctanh(x) function for array elements.
MathSignif	Rounds a value to the specified number of digits in the mantissa.
MathRank	Calculates the ranks of array elements.
MathCorrelationPearson	Calculates the Pearson's correlation coefficient.
MathCorrelationSpearman	Calculates the Spearman's correlation coefficient.
MathCorrelationKendall	Calculates the Kendall's correlation coefficient.
MathQuantile	Calculates sample quantiles corresponding to the specified probabilities.
MathProbabilityDensityEmpirical	Calculates the empirical probability density function for random values.
MathCumulativeDistributionEmpirical	Calculates the empirical cumulative distribution function for random values.

MathRandomNonZero

Returns a random number with a floating point in the range from 0.0 to 1.0.

```
double  MathRandomNonZero()
```

Return Value

Random number with a floating point in the range from 0.0 to 1.0.

MathMoments

Calculates the first 4 moments of array elements: mean, variance, skewness, kurtosis.

```
bool  MathMoments(
   const double&  array[],           // array of values
   double&        mean,              // variable for the mean
   double&        variance,          // variable for the variance
   double&        skewness,          // variable for the skewness
   double&        kurtosis,          // variable for the kurtosis
   const int      start=0,           // initial index
   const int      count=WHOLE_ARRAY  // the number of elements
   )
```

Parameters

array[]

[in] Array of values.

mean

[out] Variable for the mean (1st moment).

variance

[out] Variable for the variance (2nd moment).

skewness

[out] Variable for the skewness (3rd moment).

kurtosis

[out] Variable for the kurtosis (4th moment).

start=0

[in] Initial index for calculation.

count=WHOLE_ARRAY

[in] The number of elements for calculation.

Return Value

Returns true if the moments have been calculated successfully, otherwise false.

MathPowInt

Raises a number to the specified integer power.

```
double  MathPowInt(
   const double  x,      // value of the number
   const int     power   // power to raise to
   )
```

Parameters

x

[in] Double-precision floating-point number to be raised to the power.

power

[in] Integer specifying the power.

Return Value

The number *x*, raised to the specified power.

MathFactorial

Calculates the factorial of the specified integer.

```
double  MathFactorial(
   const int  n    // value of the number
   )
```

Parameters

n

[in] Integer number, the factorial of which is to be calculated.

Return Value

The factorial of the number.

MathTrunc

Calculates the integer part of the specified number or array elements.

Version for working with a double-precision floating-point number:

```
double  MathTrunc(
   const double    x           // value of the number
   )
```

Return Value

The integer part of the specified number.

Version for working with an array of double-precision floating-point numbers. The results are output to a new array:

```
bool  MathTrunc(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Return Value

Returns true if successful, otherwise false.

Version for working with an array of double-precision floating-point numbers. The results are output to the original array:

```
bool  MathTrunc(
   double&        array[]     // array of values
   )
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Double-precision floating-point number, the integer part of which is to be obtained.

array[]

[in] Array of double-precision floating-point numbers, the integer parts of which are to be obtained.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

MathRound

Rounds a double-precision floating-point number or an array of such numbers to the specified number of decimal places.

Version for rounding a double-precision floating-point number to the specified number of decimal places:

```
double  MathRound(
   const double   x,          // value of the number
   const int      digits      // the number of decimal places
   )
```

Return Value

A number nearest to the x parameter, with the number of decimal places equal to digits.

Version for rounding an array of double-precision floating-point numbers to the specified number of decimal places. The results are output to a new array.

```
bool  MathRound(
   const double&  array[],    // array of values
   int            digits,     // the number of decimal places
   double&        result[]    // array of results
   )
```

Return Value

Returns true if successful, otherwise false.

Version for rounding an array of double-precision floating-point numbers to the specified number of decimal places. The results are output to the original array.

```
bool  MathRound(
   double&        array[],    // array of values
   int            digits      // the number of decimal places
   )
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Double-precision floating-point number to be rounded.

digits

[in] The number of decimal places in the returned value.

array[]

[in] Array of double-precision floating-point numbers to be rounded.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

MathArctan2

Returns the arctangent of the quotient of two arguments (x, y).

Version for working with the ratio of the two specified numbers (x, y):

```
double  MathArctan2(
   const double       y,          // Y coordinate
   const double       x           // X coordinate
   )
```

Return Value

Angle θ, measured in radians, so that -π≤θ≤π and tan (θ) = y or x, where (x, y) is a point in a Cartesian coordinate system.

Version for working with the ratio of the element pairs from the x and y arrays:

```
bool  MathArctan2(
   const double&       x[],        // array of x values
   const double&       y[],        // array of y values
   double&             result[]    // array of results
   )
```

Return Value

Returns true if successful, otherwise false.

Parameters

y

[in] The Y coordinate of the point.

x

[in] The X coordinate of the point.

x[]

[in] Array of X coordinates of the points.

y[]

[in] Array of Y coordinates of the points.

result[]

[out] Array to output the results

Notes

Please note the following.

- For (x, y) in the quadrant 1, the return value will be: 0 < θ < π/2.
- For (x, y) in the quadrant 2, the return value will be: π/2 < θ≤π.
- For (x, y) in the quadrant 3, the return value will be: -π < θ <-π/2.
- For (x, y) in the quadrant 4, the return value will be: -π/2 < θ < 0.

The return value for the points outside these quadrants is indicated below.

- If y is 0 and x is not negative, then θ = 0.
- If y is 0 and x is negative, then θ = π.
- If y is a positive number, and x is 0, then θ = π/2.
- If y is negative and x is 0, then θ = -π/2.
- If y is 0 and x is 0, then θ = -π/2.

If the value of the x or y parameter is NaN, or if the values of the x and y parameters are equal to the value PositiveInfinity or NegativeInfinity, the method returns the NaN value.

MathGamma

Calculates the value of the gamma function for the real argument *x*.

```
double  MathGamma(
   const double  x       // argument of the function
   )
```

Parameters

x

[in] The real argument of the function.

Return Value

The value of the gamma function.

MathGammaLog

Calculates the logarithm of the gamma function for the real argument x.

```
double  MathGammaLog(
   const double  x      // argument of the function
   )
```

Parameters

x

[in] The real argument of the function.

Return Value

Logarithm of the function.

MathBeta

Calculates the value of the beta function for the real arguments a and b.

```
double  MathBeta(
   const double  a,      // the first argument of the function
   const double  b       // the second argument of the function
   )
```

Parameters

a

[in] The a argument of the function.

b

[in] The b argument of the function.

Return Value

Value of the function.

MathBetaLog

Calculates the logarithm of the beta function for the real arguments a and b.

```
double  MathBetaLog(
   const double  a,       // the first argument of the function
   const double  b        // the second argument of the function
   )
```

Parameters

a

[in] The a argument of the function.

b

[in] The b argument of the function.

Return Value

Logarithm of the function.

MathBetaIncomplete

Calculates the value of the incomplete beta function.

```
double  MathBetaIncomplete(
   const double  x,       // argument of the function
   const double  p,       // the first parameter of the function
   const double  q        // the second parameter of the function
   )
```

Parameters

x

[in] The argument of the function.

p

[in] The first parameter of the beta function, must be >0.0.

q

[in] The second parameter of the beta function, must be >0.0.

Return Value

Value of the function.

MathGammaIncomplete

Calculates the value of the incomplete gamma function.

```
double  MathGammaIncomplete(
   double  x,           // argument of the function
   double  alpha        // parameter of the function
   )
```

Parameters

x

[in] The argument of the function.

alpha

[in] The parameter of the incomplete gamma function.

Return Value

Value of the function.

MathBinomialCoefficient

Calculates the binomial coefficient: C(n,k)=n!/(k!*(n-k)!).

```
long  MathBinomialCoefficient(
   const int  n,       // the total number of elements
   const int  k        // the number of elements in combination
   )
```

Parameters

n

[in] The number of elements.

k

[in] The number of elements for each combination.

Return Value

The number of combinations of N choose K.

MathBinomialCoefficientLog

Calculates the logarithm of the binomial coefficient: Log(C(n,k))=Log(n!/(k!*(n-k)!))

Version for integer arguments:

```
double  MathBinomialCoefficientLog(
   const int      n,       // the total number of elements
   const int      k        // the number of elements in combination
   )
```

Version for real arguments:

```
double  MathBinomialCoefficientLog(
   const double   n,       // the total number of elements
   const double   k        // the number of elements in combination
   )
```

Parameters

n

[in] The number of elements.

k

[in] The number of elements for each combination.

Return Value

The logarithm of C(n,k).

MathHypergeometric2F2

Calculates the value of the Hypergeometric_2F2 (a, b, c, d, z) function using the Taylor's method.

```
double  MathHypergeometric2F2(
   const double  a,       // the first parameter of the function
   const double  b,       // the second parameter of the function
   const double  c,       // the third parameter of the function
   const double  d,       // the fourth parameter of the function
   const double  z        // the fifth parameter of the function
   )
```

Parameters

a

[in] The first parameter of the function.

b

[in] The second parameter of the function.

c

[in] The third parameter of the function.

d

[in] The fourth parameter of the function.

z

[in] The fifth parameter of the function.

Return Value

Value of the function.

MathSequence

Generates a sequence of values based on the following values: the first element, the last element, the step of the sequence.

Version for working with real values:

```
bool  MathSequence(
   const double  from,        // initial value
   const double  to,          // final value
   const double  step,        // step
   double&       result[]     // array of results
   )
```

Version for working with integer values:

```
bool  MathSequence(
   const int     from,        // initial value
   const int     to,          // final value
   const int     step,        // step
   int&          result[]     // array of results
   )
```

Parameters

from

[in] The first value of the sequence

to

[in] The last value of the sequence

step

[in] The step of the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathSequenceByCount

Generates a sequence of values based on the following values: the first element, the last element, the number of elements in the sequence.

Version for working with real values:

```
bool  MathSequenceByCount(
   const double  from,        // initial value
   const double  to,          // final value
   const int     count,       // count
   double&       result[]     // array of results
   )
```

Version for working with integer values:

```
bool  MathSequenceByCount(
   const int     from,        // initial value
   const int     to,          // final value
   const int     count,       // count
   int&          result[]     // array of results
   )
```

Parameters

from

[in] The first value of the sequence.

to

[in] The last value of the sequence.

count

[in] The number of elements in the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathReplicate

Generates a repeating sequence of values.

Version for working with real values:

```
bool  MathReplicate(
   const double&  array[],     // array of values
   const int      count,       // number of repetitions
   double&        result[]     // array of results
   )
```

Version for working with integer values:

```
bool  MathReplicate(
   const int&     array[],     // array of values
   const int      count,       // number of repetitions
   int&           result[]     // array of results
   )
```

Parameters

array[]

[in] Array for generating a sequence.

count

[in] The number of the array repetitions in the sequence.

result[]

[out] Array to output the sequence.

Return Value

Returns true if successful, otherwise false.

MathReverse

Generates an array of values with reverse order of elements.

Version for working with real values and with output of the results to a new array:

```
bool  MathReverse(
   const double&  array[],      // array of values
   double&        result[]      // array of results
   )
```

Version for working with integer values and with output of the results to a new array:

```
bool  MathReverse(
   const int&     array[],      // array of values
   int&           result[]      // array of results
   )
```

Version for working with real values and with output of the results to the original array.

```
bool  MathReverse(
   double&        array[]       // array of values
   )
```

Version for working with integer values and with output of the results to the original array:

```
bool  MathReverse(
   int&           array[]       // array of values
   )
```

Parameters

array[]

[in] Array of values.

array[]

[out] Output array with the reverse order of values.

result[]

[out] Output array with the reverse order of values.

Return Value

Returns true if successful, otherwise false.

MathIdentical

Compares two arrays of values and returns true if all elements match.

Version for working with arrays of real values:

```
bool  MathIdentical(
   const double&   array1[],       // the first array of values
   const double&   array2[]        // the second array of values
   )
```

Version for working with arrays of integer values:

```
bool  MathIdentical(
   const int&      array1[],       // the first array of values
   const int&      array2[]        // the second array of values
   )
```

Parameters

array1[]

[in] The first array to compare.

array2[]

[in] The second array to compare.

Return Value

Returns true if the arrays are equal, otherwise false.

MathUnique

Generates an array with unique values only.

Version for working with real values:

```
bool  MathUnique(
   const double&    array[],    // array of values
   double&          result[]    // array of results
   )
```

Version for working with integer values:

```
bool  MathUnique(
   const int&      array[],      // array of values
   int&            result[]      // array of results
   )
```

Parameters

array[]

[in] The source array.

result[]

[out] Array to output the unique values.

Return Value

Returns true if successful, otherwise false.

MathQuickSortAscending

The function for the simultaneous ascending sorting of the array[] and indices[] arrays using the QuickSort algorithm.

```
void  MathQuickSortAscending(
   double&   array[],      // array of values
   int&      indices[],    // array of indexes
   int       first,        // initial value
   int       last          // final value
   )
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

MathQuickSortDescending

The function for the simultaneous descending sorting of the array[] and indices[] arrays using the QuickSort algorithm.

```
void  MathQuickSortDescending(
   double&  array[],      // array of values
   int&     indices[],    // array of indexes
   int      first,        // initial value
   int      last          // final value
   )
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

MathQuickSort

The function for the simultaneous sorting of the array[] and indices[] arrays using the QuickSort algorithm.

```
void  MathQuickSort(
   double&  array[],      // array of values
   int&     indices[],    // array of indexes
   int      first,        // initial value
   int      last,         // final value
   int      mode          // direction
   )
```

Parameters

array[]

[in][out] Array to be sorted.

indices[]

[in][out] Array to store the indexes of the original array.

first

[in] Index of the element to start sorting from.

last

[in] Index of the element to stop sorting at.

mode

[in] Direction of sorting (>0 ascending; otherwise, descending).

MathOrder

Generates an integer array with permutation according to order of the array elements after sorting.

Version for working with an array of real values:

```
bool  MathOrder(
   const double&  array[],    // array of values
   int&           result[]    // array of results
   )
```

Version for working with an array of integer values:

```
bool  MathOrder(
   const int&     array[],    // array of values
   int&           result[]    // array of results
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array to output the sorted indexes.

Return Value

Returns true if successful, otherwise false.

MathBitwiseNot

Calculates the result of bitwise NOT operation for array elements.

Version with output of the results to a new array:

```
bool  MathBitwiseNot(
   const int&   array[],    // array of values
   int&         result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathBitwiseNot(
   int&         array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathBitwiseAnd

Calculates the result of bitwise AND operation for specified arrays.

```
bool  MathBitwiseAnd(
   const int&  array1[],  // the first array of values
   const int&  array2[],  // the second array of values
   int&        result[]   // array of results
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseOr

Calculates the result of bitwise OR operation for specified arrays.

```
bool  MathBitwiseOr(
   const int&  array1[],  // the first array of values
   const int&  array2[],  // the second array of values
   int&        result[]   // array of results
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseXor

Calculates the result of bitwise XOR operation for specified arrays.

```
bool  MathBitwiseXor(
   const int&  array1[],  // the first array of values
   const int&  array2[],  // the second array of values
   int&        result[]   // array of results
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathBitwiseShiftL

Calculates the result of bitwise SHL (bitwise shift left) operation for array elements.

Version with output of the results to a new array:

```
bool  MathBitwiseShiftL(
   const int&   array[],     // array of values
   const int    n,           // shift value
   int&         result[]     // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathBitwiseShiftL(
   int&         array[],     // array of values
   const int    n            // shift value
   )
```

Parameters

array[]

[in] Array of values.

n

[in] The number of bits to shift.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathBitwiseShiftR

Calculates the result of bitwise SHR (bitwise shift right) operation for array elements.

Version with output of the results to a new array:

```
bool  MathBitwiseShiftR(
   const int&   array[],     // array of values
   const int    n,           // shift value
   int&         result[]     // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathBitwiseShiftR(
   int&         array[],     // array of values
   const int    n            // shift value
   )
```

Parameters

array[]

[in] Array of values.

n

[in] The number of bits to shift.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeSum

Generates an array with the cumulative sums.

Version with output of the results to a new array:

```
bool   MathCumulativeSum(
   const double&   array[],    // array of values
   double&         result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool   MathCumulativeSum(
   double&         array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeProduct

Generates an array with the cumulative products.

Version with output of the results to a new array:

```
bool  MathCumulativeProduct(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCumulativeProduct(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeMin

Generates an array with the cumulative minima.

Version with output of the results to a new array:

```
bool  MathCumulativeMin(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCumulativeMin(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeMax

Generates an array with the cumulative maxima.

Version with output of the results to a new array:

```
bool  MathCumulativeMax(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCumulativeMax(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSin

Calculates the values of the sin(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathSin(
   const double&  array[],   // array of values
   double&        result[]   // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathSin(
   double&        array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCos

Calculates the values of the cos(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathCos(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCos(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTan

Calculates the values of the tan(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathTan(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathTan(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArcsin

Calculates the values of the arcsin(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArcsin(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArcsin(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArccos

Calculates the values of the arccos(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArccos(
   const double&   array[],     // array of values
   double&         result[]     // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArccos(
   double&         array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArctan

Calculates the values of the arctan(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArctan(
   const double&    array[],    // array of values
   double&          result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArctan(
   double&         array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSinPi

Calculates the values of the sin(pi*x) function for array elements.

Version with output of the results to a new array:

```
bool  MathSinPi(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathSinPi(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCosPi

Calculates the values of the cos(pi*x) function for array elements.

Version with output of the results to a new array:

```
bool  MathCosPi(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCosPi(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTanPi

Calculates the values of the tan(pi*x) function for array elements.

Version with output of the result to a new array:

```
bool  MathTanPi(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the result to the original array:

```
bool  MathTanPi(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathAbs

Calculates the absolute values of array elements.

Version with output of the results to a new array:

```
bool  MathAbs(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathAbs(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCeil

Returns the nearest larger integers for array elements.

Version with output of the results to a new array:

```
bool  MathCeil(
   const double&  array[],   // array of values
   double&        result[]   // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCeil(
   double&        array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathFloor

Returns the nearest smaller integers for array elements.

Version with output of the results to a new array:

```
bool  MathFloor(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathFloor(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSqrt

Calculates the square roots of array elements.

Version with output of the results to a new array:

```
bool  MathSqrt(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathSqrt(
   double&         array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathExp

Calculates the values of the exp(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathExp(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathExp(
   double&        array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathPow

Calculates the values of the pow(x, power) function for array elements.

Version with output of the results to a new array:

```
bool  MathPow(
   const double&  array[],    // array of values
   const double   power,      // power
   double&        result[]   // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathPow(
   double&        array[],    // array of values
   const double   power       // power
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog

Calculates the values of the log(x) function for array elements.

Version for calculating the natural logarithm with output of the results to a new array.

```
bool  MathLog(
   const double&   array[],       // array of values
   double&         result[]       // array of results
   )
```

Version for calculating the natural logarithm with output of the results to the original array.

```
bool  MathLog(
   double&         array[]        // array of values
   )
```

Version for calculating the logarithm to a specified base with output of the results to a new array.

```
bool  MathLog(
   const double&    array[],      // array of values
   const double     base,         // base of the logarithm
   double&          result[]      // array of results
   )
```

Version for calculating the logarithm to a specified base with output of the results to the original array.

```
bool  MathLog(
   double&          array[],      // array of values
   const double     base          // base of the logarithm
   )
```

Parameters

array[]

[in] Array of values.

base

[in] The base of the logarithm.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog2

Calculates the logarithm to the base 2 for the array elements.

Version with output of the results to a new array:

```
bool  MathLog2(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathLog2(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog10

Calculates the logarithm to the base 10 for the array elements.

Version with output of the results to a new array:

```
bool  MathLog10(
   const double&    array[],    // array of values
   double&          result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathLog10(
   double&         array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathLog1p

Calculates the values of the log(1+x) function for array elements.

Version with output of the results to a new array:

```
bool  MathLog1p(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathLog1p(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathDifference

Generates an array with element differences of y[i]=x[i+lag]-x[i].

Version for a single generation of an array of real values:

```
bool  MathDifference(
   const double  &array[],      // array of values
   const int     lag,           // lag
   double        &result[]      // array of results
   )
```

Version for a single generation of an array of integer values:

```
bool  MathDifference(
   const int     &array[],      // array of values
   const int     lag,           // lag
   int           &result[]      // array of results
   )
```

Version for iterated generation of an array of real values (the number of iterations is set in the input parameters):

```
bool  MathDifference(
   const double  &array[],      // array of values
   const int     lag,           // lag
   const int     differences,   // number of iterations
   double        &result[]      // array of results
   )
```

Version for iterated generation of an array of integer values (the number of iterations is set in the input parameters):

```
bool  MathDifference(
   const int&    array[],       // array of values
   const int     lag,           // lag
   const int     differences,   // number of iterations
   int&          result[]       // array of results
   )
```

Parameters

array[]

[in] Array of values.

lag

[in] Lag parameter.

differences

[in] The number of iterations.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

MathSample

Generates a random sample from the array elements.

Version for working with an array of real values:

```
bool  MathSample(
   const double&  array[],           // array of values
   const int      count,             // count
   double&        result[]           // array of results
   )
```

Version for working with an array of integer values:

```
bool  MathSample(
   const int&     array[],           // array of values
   const int      count,             // count
   int&           result[]           // array of results
   )
```

Version for working with an array of real values. It is possible to obtain a sample with replacement:

```
bool  MathSample(
   const double&  array[],           // array of values
   const int      count,             // count
   const bool     replace,           // flag
   double&        result[],          // array of results
   )
```

Version for working with an array of integer values. It is possible to obtain a sample with replacement:

```
bool  MathSample(
   const int&     array[],           // array of values
   const int      count,             // count
   const bool     replace,           // flag
   int&           result[]           // array of results
   )
```

Version for working with an array of real values, for which the probabilities of sampling are defined.

```
bool  MathSample(
   const double&  array[],           // array of values
   double&        probabilities[],   // array of probabilities
   const int      count,             // count
   double&        result[]           // array of results
   )
```

Version for working with an array of integer values, for which the probabilities of sampling are defined.

```
bool  MathSample(
   const int&      array[],            // array of values
   double&         probabilities[],    // array of probabilities
   const int       count,              // count
   int&            result[]            // array of results
   )
```

Version for working with an array of real values, for which the probabilities of sampling are defined. It is possible to obtain a sample with replacement:

```
bool  MathSample(
   const double&   array[],            // array of values
   double&         probabilities[],    // array of probabilities
   const int       count,              // count
   const bool      replace,            // flag
   double&         result[]            // array of results
   )
```

Version for working with an array of integer values, for which the probabilities of sampling are defined. It is possible to obtain a sample with replacement:

```
bool  MathSample(
   const int&      array[],            // array of values
   double&         probabilities[],    // array of probabilities
   const int       count,              // count
   const bool      replace,            // flag
   int&            result[]            // array of results
   )
```

Parameters

array[]

[in] Array of integer values.

probabilities[]

[in] Array of probabilities for sampling the elements.

count

[in] The number of elements.

replace

[in] Parameter that allows sampling with replacement.

result[]

[out] Array to output the results.

Return Value

Returns true if successful, otherwise false.

Note

The replace=true argument allows performing random sampling of the elements with replacement back to the original sequence.

MathTukeySummary

Calculates the Tukey's five-number summary (minimum, lower quartile, median, upper quartile, maximum) for the array elements.

```
bool  MathTukeySummary(
   const double&   array[],        // array of values
   const bool      removeNAN,      // flag
   double&         minimum,        // minimum value
   double&         lower_hinge,    // lower quartile
   double&         median,         // median value
   double&         upper_hinge,    // upper quartile
   double&         maximum         // maximum value
   )
```

Parameters

array[]

[in] Array of real values.

removeNAN

[in] Flag that indicates if non-numeric values are to be removed.

minimum

[out] Variable to store the minimum value.

lower_hinge

[out] Variable to store the lower quartile.

median

[out] Variable to store the median value.

upper_hinge

[out] Variable to store the upper quartile.

maximum

[out] Variable to store the maximum value.

Return Value

Returns true if successful, otherwise false.

MathRange

Calculates the minima and maxima of array elements.

```
bool  MathRange(
   const double&  array[],   // array of values
   double&        min,       // minimum value
   double&        max        // maximum value
   )
```

Parameters

array[]

[in] Array of values.

min

[out] Variable to store the minimum value.

max

[out] Variable to store the maximum value.

Return Value

Returns true if successful, otherwise false.

MathMin

Returns the minimum value of all array elements.

```
double  MathMin(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The minimum value.

MathMax

Returns the maximum value of all array elements.

```
double  MathMax(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The maximum value.

MathSum

Returns the sum of array elements.

```
double  MathSum(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The sum of the elements.

MathProduct

Returns the product of array elements.

```
double  MathProduct(
   const double&   array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The product of the elements.

MathStandardDeviation

Calculates the standard deviation of array elements.

```
double  MathStandardDeviation(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

Standard deviation.

MathAverageDeviation

The function calculates the average absolute deviation of array elements.

```
double  MathAverageDeviation(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The average absolute deviation of array elements.

MathMedian

Calculates the median value of array elements.

```
double  MathMedian(
   double&  array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The median value.

MathMean

Calculates the mean values of array elements.

```
double  MathMean(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

The mean value.

MathVariance

The function calculates the variance (second moment) of array elements.

```
double  MathVariance(
   const double&  array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

Value of the variance.

MathSkewness

The function calculates the skewness (third moment) of array elements.

```
double  MathSkewness(
   const double&  array[]   // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

Skewness.

MathKurtosis

The function calculates the kurtosis (fourth moment) of array elements.

```
double  MathKurtosis(
   const double&  array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

Return Value

Kurtosis.

MathExpm1

Calculates the values of the exp(x)-1 function for array elements.

Version with output of the results to a new array:

```
bool  MathExpm1(
   const double&    array[],    // array of values
   double&          result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathExpm1(
   double&          array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSinh

Calculates the values of the sinh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathSinh(
   const double&  array[],   // array of values
   double&        result[]   // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathSinh(
   double&        array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathCosh

Calculates the values of the cosh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathCosh(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathCosh(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathTanh

Calculates the values of the tanh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathTanh(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathTanh(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArcsinh

Calculates the values of the arcsinh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArcsinh(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArcsinh(
   double&          array[]    // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArccosh

Calculates the values of the arccosh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArccosh(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArccosh(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathArctanh

Calculates the values of the arctanh(x) function for array elements.

Version with output of the results to a new array:

```
bool  MathArctanh(
   const double&  array[],    // array of values
   double&        result[]    // array of results
   )
```

Version with output of the results to the original array:

```
bool  MathArctanh(
   double&        array[]     // array of values
   )
```

Parameters

array[]

[in] Array of values.

result[]

[out] Array of output values.

array[]

[out] Array of output values.

Return Value

Returns true if successful, otherwise false.

MathSignif

Rounds a value to the specified number of digits in the mantissa.

Version for working with a real value:

```
double  MathSignif(
   const double   x,           // value
   const int      digits       // number of decimal places
   )
```

Return Value

The rounded value.

Version for working with an array of real values and with output of the results to a separate array:

```
bool  MathSignif(
   const double&  array[],    // array of values
   int            digits,     // number of decimal places
   double         result[]    // array of results
   )
```

Return Value

Returns true if successful, otherwise false.

Version for working with an array of real values and with output of the results to the original array:

```
bool  MathSignif(
   double&        array[],    // array of values
   int            digits      // number of decimal places
   )
```

Return Value

Returns true if successful, otherwise false.

Parameters

x

[in] Real value to be rounded.

digits

[in] Number of decimal places.

array[]

[in] Array of real values.

array[]

[out] Array of output values.

result[]

[out] Array of output values.

MathRank

Calculates the ranks of array elements.

Version for working with an array of real values:

```
bool  MathRank(
   const double&  array[],   // array of values
   double&        rank[]     // array of ranks
   )
```

Version for working with an array of integer values:

```
bool  MathRank(
   const int&     array[],   // array of values
   double&        rank[]     // array of ranks
   )
```

Parameters

array[]

[in] Array of values.

rank[]

[out] Array to output the ranks.

Return Value

Returns true if successful, otherwise false.

MathCorrelationPearson

Calculates the Pearson's correlation coefficient.

Version for working with arrays of real values:

```
bool  MathCorrelationPearson(
   const double&   array1[],   // the first array of values
   const double&   array2[],   // the second array of values
   double&         r           // correlation coefficient
   )
```

Version for working with arrays of integer values:

```
bool  MathCorrelationPearson(
   const int&      array1[],   // the first array of values
   const int&      array2[],   // the second array of values
   double&         r           // correlation coefficient
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

r

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathCorrelationSpearman

Calculates the Spearman's correlation coefficient.

Version for working with arrays of real values:

```
bool  MathCorrelationSpearman(
   const double&   array1[],   // the first array of values
   const double&   array2[],   // the second array of values
   double&         r           // correlation coefficient
   )
```

Version for working with arrays of integer values:

```
bool  MathCorrelationSpearman(
   const int&      array1[],    // the first array of values
   const int&      array2[],    // the second array of values
   double&         r            // correlation coefficient
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

r

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathCorrelationKendall

Calculates the Kendall's correlation coefficient.

Version for working with arrays of real values:

```
bool  MathCorrelationKendall(
   const double&   array1[],   // the first array of values
   const double&   array2[],   // the second array of values
   double&         tau         // correlation coefficient
   )
```

Version for working with arrays of integer values:

```
bool  MathCorrelationKendall(
   const int&      array1[],   // the first array of values
   const int&      array2[],   // the second array of values
   double&         tau         // correlation coefficient
   )
```

Parameters

array1[]

[in] The first array of values.

array2[]

[in] The second array of values.

tau

[out] Variable to store the correlation coefficient.

Return Value

Returns true if successful, otherwise false.

MathQuantile

Calculates sample quantiles corresponding to the specified probabilities: Q[i](p) = (1 - gamma) *x[j] + gamma*x[j+1]

```
bool  MathQuantile(
   const double&   array[],      // array of values
   const double&   probs[],      // array of probabilities
   double&         quantile[]    // array to output the quantiles
   )
```

Parameters

array[]

[in] Array of values.

probs[]

[in] Array of probabilities.

quantile[]

[out] Array to output the quantiles.

Return Value

Returns true if successful, otherwise false.

MathProbabilityDensityEmpirical

The function calculates the empirical probability density function (pdf) for random values from an array.

```
bool  MathProbabilityDensityEmpirical(
   const double&  array[],    // array of random values
   const int      count,      // the number of pairs
   double&        x[],        // array of x values
   double&        pdf[]       // array of pdf values
   )
```

Parameters

array[]

[in] Array of random values.

count

[in] The number of (x, pdf(x)) pairs.

x[]

[out] Array to output the x values.

pdf[]

[out] Array to output the pdf(x) values.

Return Value

Returns true if successful, otherwise false.

MathCumulativeDistributionEmpirical

The function calculates the empirical cumulative distribution function (cdf) for random values from an array.

```
bool  MathCumulativeDistributionEmpirical(
   const double&  array[],    // array of random values
   const int      count,      // the number of pairs
   double&        x[],        // array of x values
   double&        cdf[]       // array of cdf values
   )
```

Parameters

array[]

[in] Array of random values.

count

[in] The number of (x, cdf(x)) pairs.

x[]

[out] Array to output the x values.

cdf[]

[out] Array to output the cdf(x) values.

Return Value

Returns true if successful, otherwise false.

Fuzzy is a library for working with fuzzy logic

Fuzzy logic is a synthesis of the traditional Aristotelian logic when truth is marked as a linguistic variable. Fuzzy logic, equivalent to classical logic, has its own fuzzy logic operations on fuzzy sets defined. There are the same operations for fuzzy sets as well as for ordinary sets, only their calculation is by far more difficult. We should also note that the composition of fuzzy sets constitutes as a fuzzy set.

The main principles of fuzzy logic, setting it apart from classical logic, are the maximum proximity to the reflection of reality and a high level of subjectivity, which can lead to significant errors in calculations.

Fuzzy model (or system) is a mathematical model whose calculation is based on fuzzy logic. Construction of such models is applicable when the subject of study has a weak formalization and its exact mathematical description is too complex or unknown. The quality of these models' output values (error model) directly depends only on the Expert Advisor, which set up this model. The best option to minimize errors is to draw the most complete and comprehensive model and subsequently adjust it with machine learning on a large training set.

The progress of a model construction can be divided into three main stages:

1. Definition of input and output characteristics of a model.
2. Building a knowledge base.
3. Selecting one of the methods of fuzzy inference (Mamdani or Sugeno).

The first stage directly effects the consequent two and determines the future operation of the model. A knowledge base or, as sometimes called, rule base is a set of fuzzy rules type: "if, then" that define the relationship between inputs and outputs of the examined object. The number of rules in the system is not limited and is also determined by the Expert Advisor. The generalized format of fuzzy rules is as follows:

If rule condition, then rule conclusion.

Rule condition describes the current state of the object, and rule conclusion — how this condition affects the object. General view of conditions and conclusions cannot be selected because they are determined by a fuzzy inference.

Each rule in the system has its weight — this characteristic defines the importance of a rule in the model. Weighting factors are assigned to a rule within range [0, 1]. In many examples with fuzzy models, which can be found in the relevant literature, weight data is not specified, but it does not mean that it is not present. In fact, in such case for each rule from the database, the weight is fixed and equals unity. There can be two types of terms and conclusions for each rule:

1. simple — includes one fuzzy variable;
2. complex — includes several fuzzy variables.

Depending on the created knowledge base, the system of fuzzy inference is determined for a model. Fuzzy logical inference is a receipt of conclusion in form of a fuzzy set corresponding to the current value of the inputs with use of knowledge base and fuzzy operations. The two main types of fuzzy inference are Mamdani and Sugeno.

Membership functions

A **membership function** is a function that allows to calculate the membership degree of a random element of the universal set to a fuzzy set. Consequently, the domain of a membership function should be within the range [0, 1].

In most cases, the membership function is continuous and monotonic.

Classes of membership functions	Description
CConstantMembershipFunction	Class for implementing a membership function as a straight line in parallel with the coordinate axis
CCompositeMembershipFunction	Class for implementing a composition of membership functions
CDifferencTwoSigmoidalMembershipFunction	Class for implementing the membership function in the form of a difference between two sigmoid functions with the A1, A2, C1 and C2 parameters
CGeneralizedBellShapedMembershipFunction	Class for implementing a generalized bell-shaped membership function with A, B and C parameters
CNormalCombinationMembershipFunction	Class for implementing a two-sided Gaussian membership function with the B1, B2, Sigma1 and Sigma2 parameters
CNormalMembershipFunction	Class for implementing a symmetrical Gaussian membership function with the B and Sigma parameters
CP_ShapedMembershipFunction	Class for implementing a pi-shaped membership function with the A, B, C and D parameters
CProductTwoSigmoidalMembershipFunction	Class for implementing the membership function in the form of a product of two sigmoid functions with the A1, A2, C1 and C2 parameters
CS_ShapedMembershipFunction	Class for implementing an S-like membership function with the A and B parameters
CTrapezoidMembershipFunction	Class for implementing a trapezoidal membership function with the X1, X2, X3 and X4 parameters
CTriangularMembershipFunction	Class for implementing a triangle membership function with the X1, X2 and X3 parameters
CSigmoidalMembershipFunction	Class for implementing a sigmoid membership function with the A and C parameters

Classes of membership functions	Description
CZ_ShapedMembershipFunction	Class for implementing a z-like membership function with the A and B parameters.
IMembershipFunction	Basic class for all membership function classes.

CConstantMembershipFunction

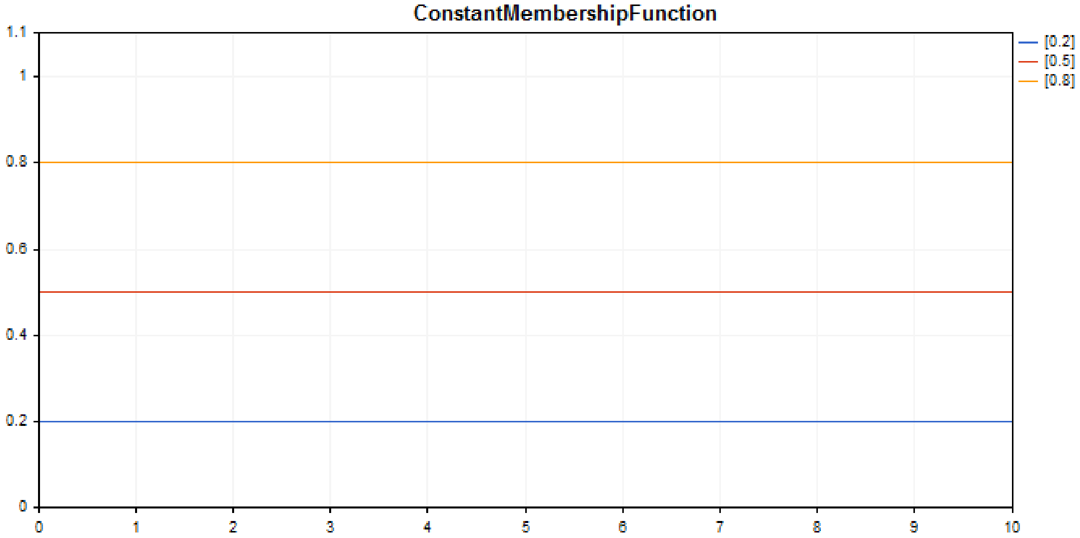
Class for implementing a membership function as a straight line in parallel with the coordinate axis.

Description

The function is described by the equation:

y(x)=c

Therefore, the membership degree for the function is the same along the entire numerical axis and is equal to the parameter specified in the constructor.



A sample code for plotting a chart is displayed below.

Declaration

```
class CConstantMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CConstantMembershipFunction

Class methods

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                ConstantMembershipFunction.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                           https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CConstantMembershipFunction func1(0.2);
CConstantMembershipFunction func2(0.5);
CConstantMembershipFunction func3(0.8);
//--- Create wrappers for membership functions
double ConstantMembershipFunction1(double x) { return(func1.GetValue(x)); }
double ConstantMembershipFunction2(double x) { return(func2.GetValue(x)); }
double ConstantMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"ConstantMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"ConstantMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("ConstantMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(ConstantMembershipFunction1,0.0,10.0,1.0,CURVE_LINES,"[0.2]");
   graphic.CurveAdd(ConstantMembershipFunction2,0.0,10.0,1.0,CURVE_LINES,"[0.5]");
   graphic.CurveAdd(ConstantMembershipFunction3,0.0,10.0,1.0,CURVE_LINES,"[0.8]");
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
```

```
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const double  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

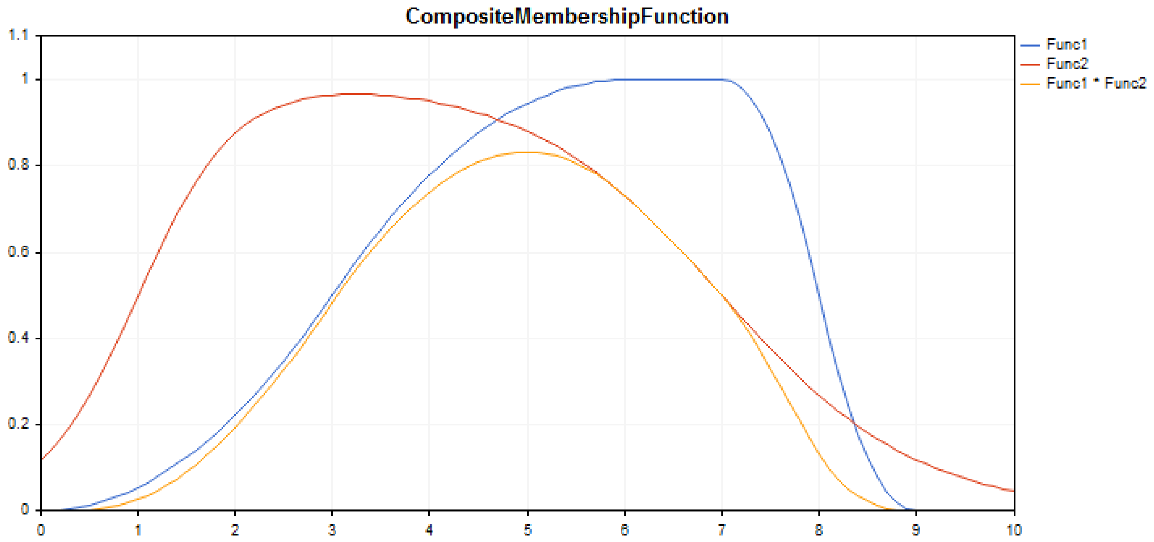
Membership function value.

CCompositeMembershipFunction

Class for implementing a composition of membership functions.

Description

Composition of membership functions is a combination of two or more membership functions using a specified operator.



A sample code for plotting a chart is displayed below.

Declaration

```
class CCompositeMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CCompositeMembershipFunction

Class methods

Class method	Description
CompositionType	Sets the composition operator.
MembershipFunctions	Gets the list of membership functions.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                 CompositeMembershipFunction.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CP_ShapedMembershipFunction func2(0,6,7,9);
CCompositeMembershipFunction composite(ProdMF,GetPointer(func1),GetPointer(func2));
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions(double x) { return(func1.GetValue(x)); 
double P_ShapedMembershipFunction(double x) { return(func2.GetValue(x)); }
double CompositeMembershipFunction(double x) { return(composite.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"CompositeMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"CompositeMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("CompositeMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(P_ShapedMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1");
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions,0.0,10.0,0.1,CURVE_LINES,"
   graphic.CurveAdd(CompositeMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1 * Fun
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
```

```
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

CompositionType

Sets the composition operator.

```
void  CompositionType(
   MfCompositionType  value      // operator type
   )
```

Parameters

value

[in] Composition operator type.

Note

The following operator types are available:

- MinMF (minimum of functions)
- MaxMF (maximum of functions)
- ProdMF (product of functions)
- SumMF (sum of functions)

MembershipFunctions

Gets the list of membership functions included into a composition.

```
CList*  MembershipFunctions(
   void         // list of membership functions
   )
```

Return Value

List of membership functions.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x      // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

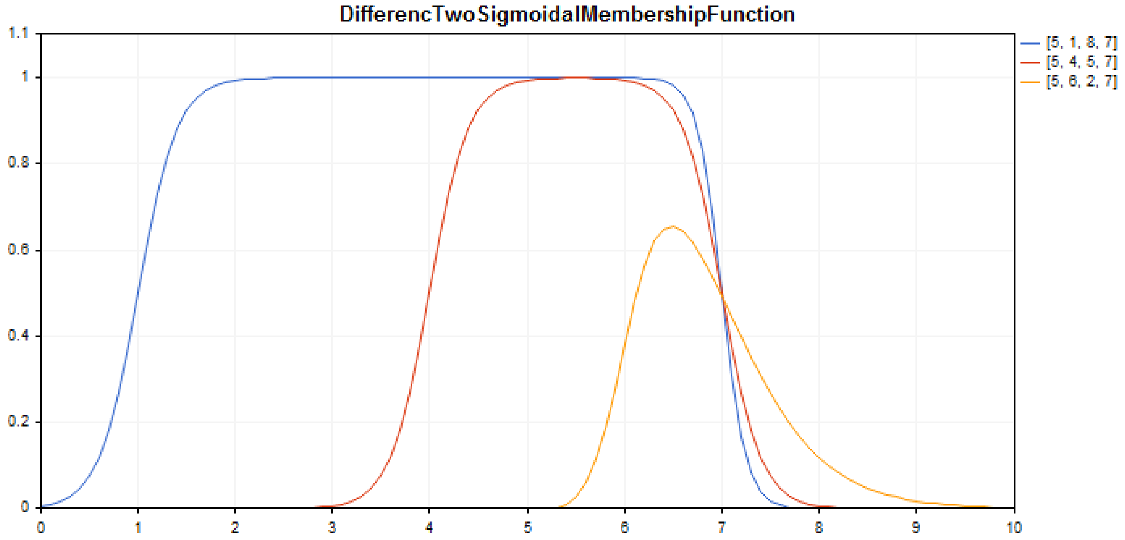
Membership function value.

CDifferencTwoSigmoidalMembershipFunction

Class for implementing the membership function in the form of a difference between two sigmoid functions with the A1, A2, C1 and C2 parameters.

Description

The function is based on a sigmoid curve. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



[A sample code](#) for plotting a chart is displayed below.

Declaration

```
class CDifferencTwoSigmoidalMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CDifferencTwoSigmoidalMembershipFunction

Class methods

Class method	Description
A1	Gets and sets the first membership function slope ratio.

Class method	Description
A2	Gets and sets the second membership function slope ratio.
C1	Gets and sets the first membership function inflection coordinate parameter.
C2	Gets and sets the second membership function inflection coordinate parameter.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                      DifferencTwoSigmoidalMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CDifferencTwoSigmoidalMembershipFunction func1(5,1,8,7);
CDifferencTwoSigmoidalMembershipFunction func2(5,4,5,7);
CDifferencTwoSigmoidalMembershipFunction func3(5,6,2,7);
//--- Create wrappers for membership functions
double DifferencTwoSigmoidalMembershipFunction1(double x) { return(func1.GetValue(x));
double DifferencTwoSigmoidalMembershipFunction2(double x) { return(func2.GetValue(x));
double DifferencTwoSigmoidalMembershipFunction3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"DifferencTwoSigmoidalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"DifferencTwoSigmoidalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("DifferencTwoSigmoidalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A1 (Get method)

Gets the first membership function slope ratio.

```
double  A1()
```

Return Value

Slope ratio value.

A1 (Set method)

Sets the first membership function slope ratio.

```
void  A1(
   const double  a1       // slope ratio value
   )
```

Parameters

a1

[in] Slope ratio value.

A2 (Get method)

Gets the second membership function slope ratio.

```
double  A2()
```

Return Value

Slope ratio value.

A2 (Set method)

Sets the second membership function slope ratio.

```
void  A2(
   const double  a2       // slope ratio value
   )
```

Parameters

a2

[in] Slope ratio value.

C1 (Get method)

Gets the first membership function inflection coordinate parameter.

```
double  C1()
```

Return Value

Inflection coordinate value.

C1 (Set method)

Sets the first membership function inflection coordinate parameter.

```
void  C1(
   const double  c1       // inflection coordinate value
   )
```

Parameters

c1

[in] Inflection coordinate value.

C2 (Get method)

Gets the second membership function inflection coordinate parameter.

```
double  C2()
```

Return Value

Inflection coordinate value.

C2 (Set method)

Sets the second membership function inflection coordinate parameter.

```
void  C2(
   const double  c2       // inflection coordinate value
   )
```

Parameters

c2

[in] Inflection coordinate value.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const double   x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

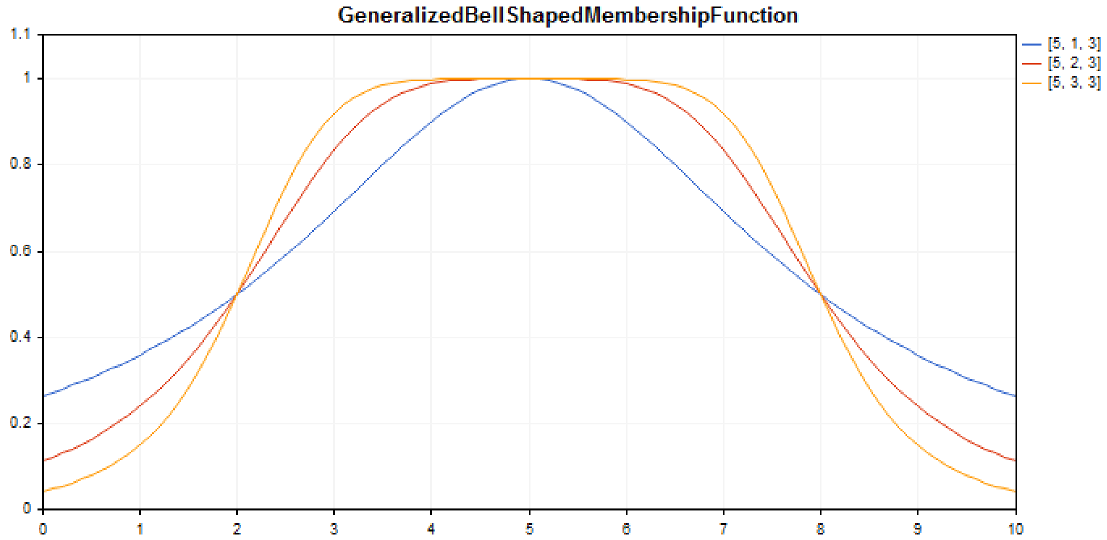
Membership function value.

CGeneralizedBellShapedMembershipFunction

Class for implementing a generalized bell-shaped membership function with A, B and C parameters.

Description

Generalized bell-shaped membership function shape is similar to Gaussian functions. The function is smooth and takes non-zero values along the entire definition area.



A sample code for plotting a chart is displayed below.

Declaration

```
class CGeneralizedBellShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CGeneralizedBellShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the membership function concentration ratio.
B	Gets and sets the membership function slope ratio.

Class method	Description
C	Gets and sets the membership function maximum coordinate.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                      GeneralizedBellShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CGeneralizedBellShapedMembershipFunction func1(5, 1, 3);
CGeneralizedBellShapedMembershipFunction func2(5, 2, 3);
CGeneralizedBellShapedMembershipFunction func3(5, 3, 3);
//--- Create wrappers for membership functions
double GeneralizedBellShapedMembershipFunction1(double x) { return(func1.GetValue(x));
double GeneralizedBellShapedMembershipFunction2(double x) { return(func2.GetValue(x));
double GeneralizedBellShapedMembershipFunction3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"GeneralizedBellShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"GeneralizedBellShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("GeneralizedBellShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
   graphic.CurveAdd(GeneralizedBellShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A (Get method)

Gets the membership function concentration ratio.

```
double  A()
```

Return Value

Concentration ratio value.

A (Set method)

Sets the membership function concentration ratio.

```
void  A(
   const double  a       // concentration ratio value
   )
```

Parameters

a

[in] Membership function concentration ratio.

B (Get method)

Gets the membership function slope ratio.

```
double  B()
```

Return Value

Slope ratio value.

B (Set method)

Sets the membership function slope ratio.

```
void  B(
   const double  b       // slope ratio value
   )
```

Parameters

b

[in] Membership function slope ratio.

C (Get method)

Gets the membership function maximum coordinate.

```
double  C()
```

Return Value

Membership function maximum coordinate.

C (Set method)

Sets the membership function maximum coordinate.

```
void  C(
   const double  c       // maximum coordinate value
   )
```

Parameters

c

[in] Membership function maximum coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

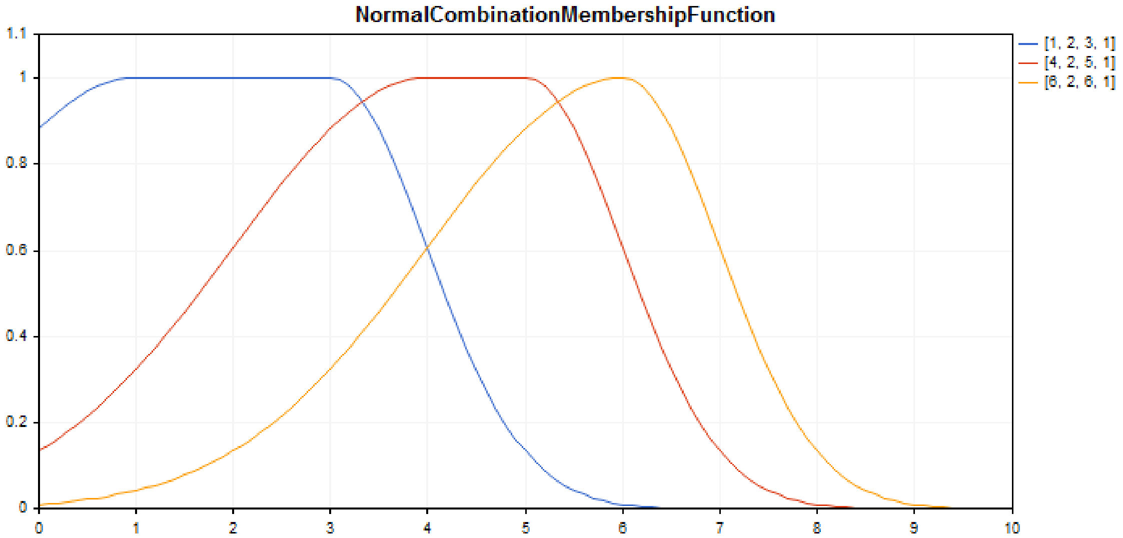
Membership function value.

CNormalCombinationMembershipFunction

Class for implementing a two-sided Gaussian membership function with the B1, B2, Sigma1 and Sigma2 parameters.

Description

The two-sided Gaussian membership function is formed using Gaussian distribution. It allows setting asymmetrical membership functions. The function is smooth and takes non-zero values along the entire definition area.



A sample code for plotting a chart is displayed below.

Declaration

```
class CNormalCombinationMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CNormalCombinationMembershipFunction

Class methods

Class method	Description
B1	Gets and sets the value of the first membership function center.

Class method	Description
B2	Gets and sets the value of the second membership function center.
Sigma1	Gets and sets the first parameter of the membership function curvature.
Sigma2	Gets and sets the second parameter of the membership function curvature.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                          NormalCombinationMembershipFunction.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalCombinationMembershipFunction func1(1,2,3,1);
CNormalCombinationMembershipFunction func2(4,2,5,1);
CNormalCombinationMembershipFunction func3(6,2,6,1);
//--- Create wrappers for membership functions
double NormalCombinationMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalCombinationMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalCombinationMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"NormalCombinationMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"NormalCombinationMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("NormalCombinationMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(NormalCombinationMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[1,
   graphic.CurveAdd(NormalCombinationMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[4,
   graphic.CurveAdd(NormalCombinationMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[6,
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

B1 (Get method)

Gets the value of the first membership function center.

```
double  B1()
```

Return Value

Value of the first membership function center.

B1 (Set method)

Sets the value of the first membership function center.

```
void  B1(
   const double  b1       // first center value
   )
```

Parameters

b

[in] Value of the first membership function center.

B2 (Get method)

Gets the value of the second membership function center.

```
double  B2()
```

Return Value

The value of the second membership function center.

B2 (Set method)

Sets the value of the second membership function center.

```
void  B2(
   const double  b2      // second center value
   )
```

Parameters

b2

[in] Value of the second membership function center.

Sigma1 (Get method)

Gets the first parameter of the membership function curvature.

```
double  Sigma1()
```

Return Value

The value of the first parameter of the membership function curvature.

Sigma1 (Set method)

Sets the value of the first parameter of the membership function curvature.

```
void  Sigma1(
   const double  sigma1      // first curvature parameter value
   )
```

Parameters

sigma1

[in] The first parameter of the membership function curvature.

Sigma2 (Get method)

Gets the second parameter of the membership function curvature.

```
double  Sigma2()
```

Return Value

The value of the second parameter of the membership function curvature.

Sigma2 (Set method)

Sets the value of the second parameter of the membership function curvature.

```
void  Sigma2(
   const double  sigma2      // second curvature parameter value
   )
```

Parameters

sigma2

[in] The second curvature parameter of the membership function.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x      // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

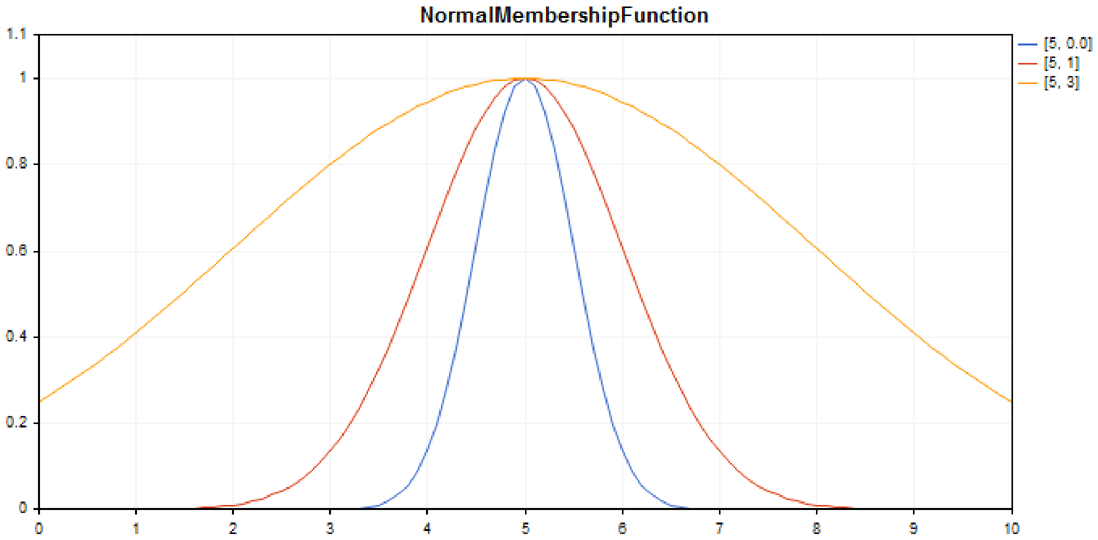
Membership function value.

CNormalMembershipFunction

Class for implementing a symmetrical Gaussian membership function with the B and Sigma parameters.

Description

The symmetrical Gaussian membership function is formed using Gaussian distribution. The function is smooth and takes non-zero values along the entire definition area.



A sample code for plotting a chart is displayed below.

Declaration

```
class CNormalMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CNormalMembershipFunction

Class methods

Class method	Description
B	Gets and sets the membership function center.
Sigma	Gets and sets the parameter of the membership function curvature.

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                   NormalMembershipFunction.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalMembershipFunction func1(5,0.5);
CNormalMembershipFunction func2(5,1);
CNormalMembershipFunction func3(5,3);
//--- Create wrappers for membership functions
double NormalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"NormalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"NormalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("NormalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(NormalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[5, 0.0]");
   graphic.CurveAdd(NormalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[5, 1]");
   graphic.CurveAdd(NormalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[5, 3]");
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
```

```
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

B (Get method)

Gets the membership function center.

```
double  B()
```

Return Value

The value of the membership function center.

B (Set method)

Sets the value of the membership function center.

```
void  B(
   const double  b       // function center value
   )
```

Parameters

b

[in] Value of the membership function center.

Sigma (Get method)

Gets the parameter of the membership function curvature.

```
double  Sigma()
```

Return Value

The value of the membership function curvature parameter.

Sigma (Set method)

Sets the value of the membership function curvature parameter.

```
void  Sigma(
   const double  sigma       // curvature parameter value
   )
```

Parameters

sigma

[in] Membership function curvature parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const double  x       // argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

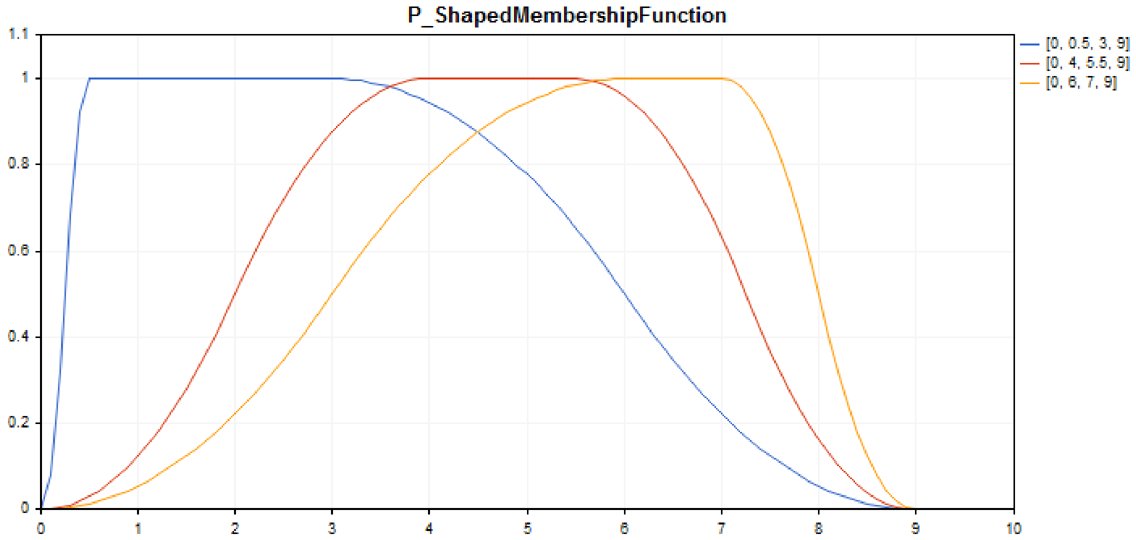
Membership function value.

CP_ShapedMembershipFunction

Class for implementing a pi-shaped membership function with the A, B, C and D parameters.

Description

The pi-shaped membership function has the form of a curvilinear trapezoid. The function is used to set asymmetric membership functions with a smooth transition from pessimistic to optimistic fuzzy number assessment.



A sample code for plotting a chart is displayed below.

Declaration

```
class CP_ShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

CObject

IMembershipFunction

CP_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the fuzzy set beginning.
B	Gets and sets the first parameter of the fuzzy set core.

Class method	Description
C	Gets and sets the second parameter of the fuzzy set core.
D	Gets and sets the parameter of the fuzzy set end.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                   P_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CP_ShapedMembershipFunction func1(0,0.5,3,9);
CP_ShapedMembershipFunction func2(0,4,5.5,9);
CP_ShapedMembershipFunction func3(0,6,7,9);
//--- Create wrappers for membership functions
double P_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double P_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"P_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"P_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("P_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(P_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 0.5, 3,
   graphic.CurveAdd(P_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 4, 5.5,
   graphic.CurveAdd(P_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[0, 6, 7, 9]
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A (Get method)

Gets the parameter of the fuzzy set beginning.

```
double  A()
```

Return Value

Parameter of the fuzzy set beginning.

A (Set method)

Sets the parameter of the fuzzy set beginning.

```
void  A(
   const double   a       // parameter of the fuzzy set beginning
   )
```

Parameters

a

[in] Parameter of the fuzzy set beginning.

B (Get method)

Gets the first parameter of the fuzzy set core.

```
double  B()
```

Return Value

The first parameter of the fuzzy set core.

B (Set method)

Sets the first parameter of the fuzzy set core.

```
void  B(
   const double   b       // value of the fuzzy set core first parameter
   )
```

Parameters

b

[in] The first parameter of the fuzzy set core.

C (Get method)

Gets the second parameter of the fuzzy set core.

```
double  C()
```

Return Value

The second parameter of the fuzzy set core.

C (Set method)

Sets the second parameter of the fuzzy set core.

```
void  C(
   const double   c       // value of the fuzzy set core second parameter
   )
```

Parameters

c

[in] The second parameter of the fuzzy set core.

D (Get method)

Gets the parameter of the fuzzy set end.

```
double  D()
```

Return Value

Value of the fuzzy set end parameter.

D (Set method)

Sets the parameter of the fuzzy set end.

```
void  D(
   const double   d       // value of the fuzzy set end parameter
   )
```

Parameters

d

[in] Value of the fuzzy set end parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const double   x
   )
```

Parameters

x

[in] membership function argument

Return Value

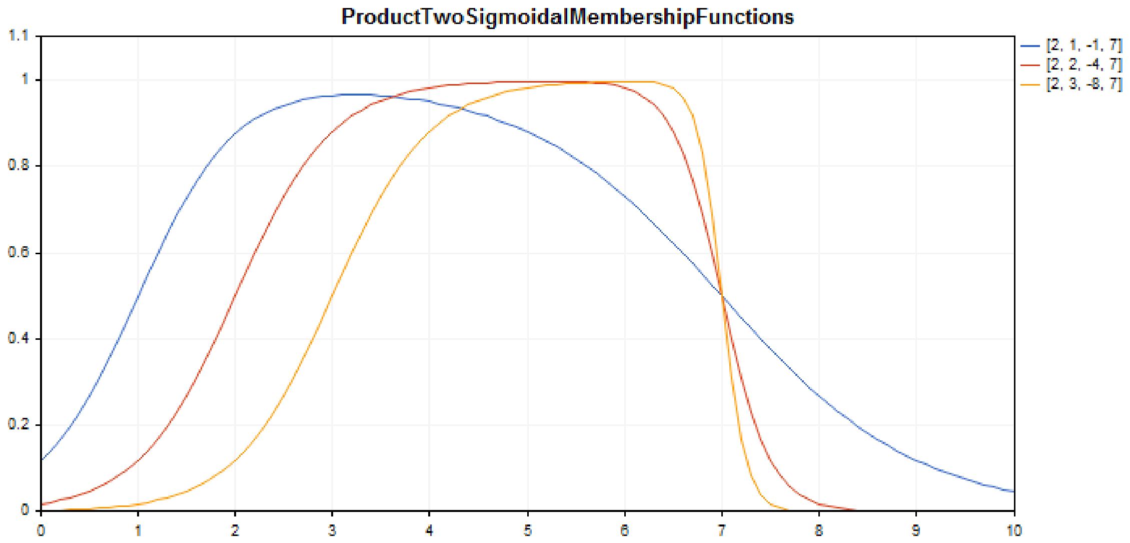
membership function value

CProductTwoSigmoidalMembershipFunction

Class for implementing the membership function in the form of a product of two sigmoid functions with the A1, A2, C1 and C2 parameters.

Description

A product of two sigmoid membership functions is applied for setting smooth asymmetric functions. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



A sample code for plotting a chart is displayed below.

Declaration

```
class CProductTwoSigmoidalMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CProductTwoSigmoidalMembershipFunctions

Class methods

Class method	Description
A1	Gets and sets the first membership function slope ratio.

Class method	Description
A2	Gets and sets the second membership function slope ratio.
C1	Gets the first membership function inflection coordinate parameter.
C2	Gets the second membership function inflection coordinate parameter.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                   ProductTwoSigmoidalMembershipFunctions.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CProductTwoSigmoidalMembershipFunctions func2(2,2,-4,7);
CProductTwoSigmoidalMembershipFunctions func3(2,3,-8,7);
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions1(double x) { return(func1.GetValue(x));
double ProductTwoSigmoidalMembershipFunctions2(double x) { return(func2.GetValue(x));
double ProductTwoSigmoidalMembershipFunctions3(double x) { return(func3.GetValue(x));
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"ProductTwoSigmoidalMembershipFunctions",0,30,30,780,380))
     {
      graphic.Attach(0,"ProductTwoSigmoidalMembershipFunctions");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("ProductTwoSigmoidalMembershipFunctions");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions1,0.0,10.0,0.1,CURVE_LINES,'
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions2,0.0,10.0,0.1,CURVE_LINES,'
   graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions3,0.0,10.0,0.1,CURVE_LINES,'
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A1 (Get method)

Gets the first membership function slope ratio.

```
double  A1()
```

Return Value

The first membership function slope ratio.

A1 (Set method)

Sets the first membership function slope ratio.

```
void  A1(
   const double  a1      // the first membership function slope ratio
   )
```

Parameters

a1

[in] The first membership function slope ratio.

A2 (Get method)

Gets the second membership function slope ratio.

```
double  A2()
```

Return Value

The second membership function slope ratio.

A2 (Set method)

Sets the second membership function slope ratio.

```
void  A2(
   const double  a2      // the second membership function slope ratio
   )
```

Parameters

a2

[in] The second membership function slope ratio.

C1 (Get method)

Gets the first membership function inflection coordinate parameter.

```
double  C1()
```

Return Value

The first membership function inflection coordinate.

C1 (Set method)

Sets the first membership function inflection coordinate.

```
void  C1(
   const double  c1      // the first membership function inflection coordinate
   )
```

Parameters

c1

[in] The first membership function inflection coordinate.

C2 (Get method)

Gets the second membership function inflection coordinate parameter.

```
double  C2()
```

Return Value

The second membership function inflection coordinate.

C2 (Set method)

Sets the second membership function inflection coordinate.

```
void  C2(
   const double  c2      // the second membership function inflection coordinate
   )
```

Parameters

c2

[in] The second membership function inflection coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

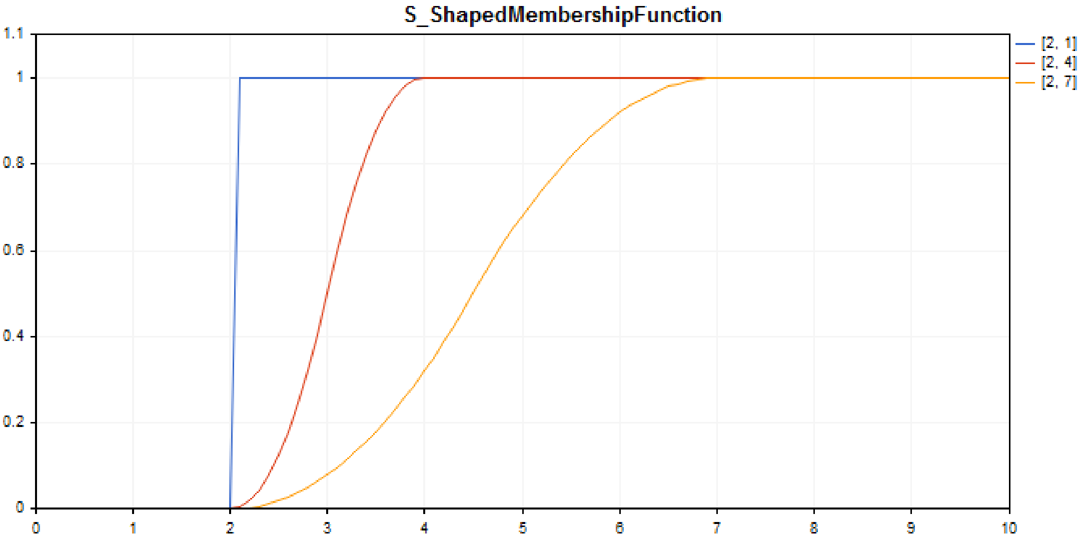
CS_ShapedMembershipFunction

Class for implementing an S-like membership function with the A and B parameters.

Description

The function sets an S-like two-parameter membership function. This is a non-decreasing function that takes values from 0 to 1. The A and B parameters define the interval, within which the function increases in non-linear trajectory from 0 to 1.

The function represents fuzzy sets of "very high" type (i.e. non-decreasing membership functions with saturation are set).



A sample code for plotting a chart is displayed below.

Declaration

```
class CS_ShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CS_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the increasing interval start.

Class method	Description
B	Gets and sets the first parameter of the fuzzy set core.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                  S_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CS_ShapedMembershipFunction func1(2,1);
CS_ShapedMembershipFunction func2(2,4);
CS_ShapedMembershipFunction func3(2,7);
//--- Create wrappers for membership functions
double S_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double S_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double S_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"S_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"S_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("S_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(S_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
   graphic.CurveAdd(S_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
   graphic.CurveAdd(S_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 7]");
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A (Get method)

Gets the parameter of the increasing interval start.

```
double  A()
```

Return Value

Increasing interval start parameter.

A (Set method)

Sets the parameter of the increasing interval start.

```
void  A(
   const double  a       // increasing interval start parameter
   )
```

Parameters

a

[in] Increasing interval start parameter.

B (Get method)

Gets the first parameter of the fuzzy set core.

```
double  B()
```

Return Value

The first parameter of the fuzzy set core.

B (Set method)

Sets the first parameter of the fuzzy set core.

```
void  B(
   const double  b       // the first parameter of the fuzzy set core
   )
```

Parameters

b

[in] The first parameter of the fuzzy set core.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

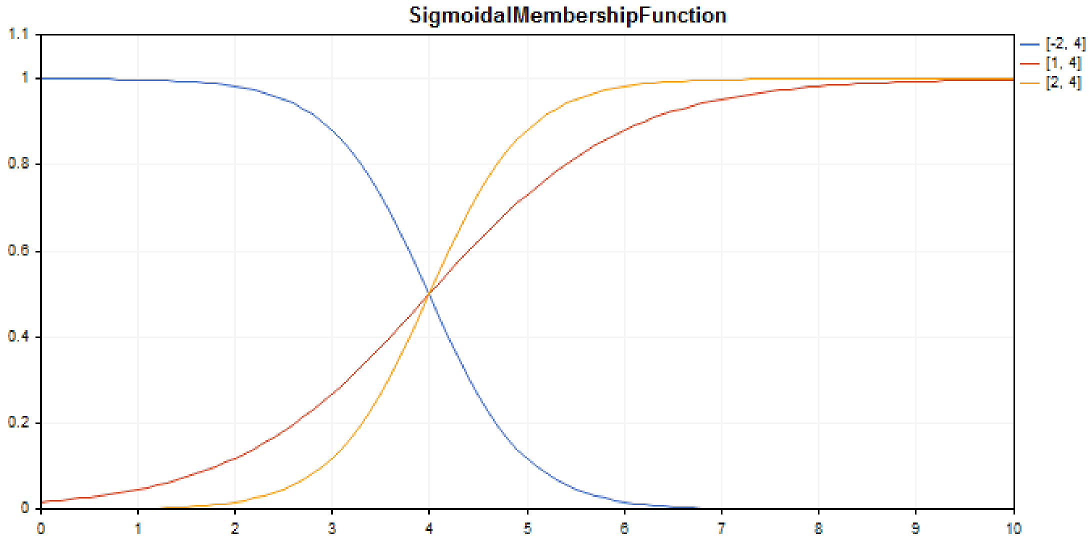
Membership function value.

CSigmoidalMembershipFunction

Class for implementing a sigmoid membership function with the A and C parameters.

Description

The sigmoid function is applied when setting monotonous membership functions. It allows creating membership functions with the values equal to 1 beginning with an argument value. Such functions are suitable if you need to set such linguistic terms as "short" or "long".



A sample code for plotting a chart is displayed below.

Declaration

```
   class CSigmoidalMembershipFuncion : public IMembershipFunction
```

Title

```
   #include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

- CObject
  - IMembershipFunction
    - CSigmoidalMembershipFunction

Class methods

Class method	Description
A	Gets and sets the membership function slope ratio.
C	Gets and sets the membership function inflection coordinate parameter.

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                 SigmoidalMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CSigmoidalMembershipFunction func1(-2, 4);
CSigmoidalMembershipFunction func2(1, 4);
CSigmoidalMembershipFunction func3(2, 4);
//--- Create wrappers for membership functions
double SigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double SigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double SigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"SigmoidalMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"SigmoidalMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("SigmoidalMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(SigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[-2, 4]");
   graphic.CurveAdd(SigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[1, 4]");
   graphic.CurveAdd(SigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
```

```
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A (Get method)

Gets the membership function slope ratio.

```
double  A()
```

Return Value

The membership function slope ratio.

A (Set method)

Sets the membership function slope ratio.

```
void  A(
   const double  a       // the first membership function slope ratio
   )
```

Parameters

a

[in] Membership function slope ratio.

C (Get method)

Gets the membership function inflection coordinate parameter.

```
double  C()
```

Return Value

Membership function inflection coordinate.

C (Set method)

Sets the membership function inflection coordinate.

```
void  C(
   const double  c       // membership function inflection coordinate
   )
```

Parameters

c

[in] The membership function inflection coordinate.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

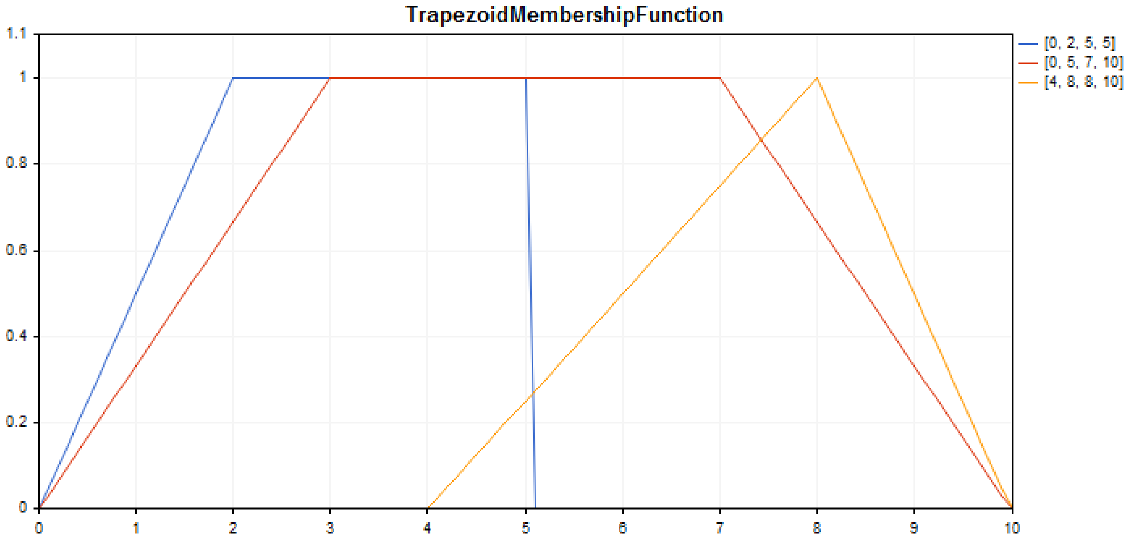
CTrapezoidMembershipFunction

Class for implementing a trapezoidal membership function with the X1, X2, X3 and X4 parameters.

Description

The function is formed using piecewise linear approximation. This is a generalization of the triangular function allowing you to assign a fuzzy set core as an interval. Such a membership function makes it possible to conveniently interpret optimistic/pessimistic assessments.

The function is used to set asymmetric membership functions of the variables with their most critical values defined within a certain interval.



A sample code for plotting a chart is displayed below.

Declaration

```
class CTrapezoidMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

CObject

  IMembershipFunction

    CTrapezoidMembershipFunction

Class methods

Class method	Description
X1	Gets and sets the value of the first point on the X axis.

Class method	Description
X2	Gets and sets the value of the second point on the X axis.
X3	Gets and sets the value of the third point on the X axis.
X4	Gets and sets the value of the fourth point on the X axis.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                TrapezoidMembershipFunction.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTrapezoidMembershipFunction func1(0,2,5,5);
CTrapezoidMembershipFunction func2(0,3,7,10);
CTrapezoidMembershipFunction func3(4,8,8,10);
//--- Create wrappers for membership functions
double TrapezoidMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TrapezoidMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TrapezoidMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"TrapezoidMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"TrapezoidMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("TrapezoidMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(TrapezoidMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5, 
   graphic.CurveAdd(TrapezoidMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 7, 
   graphic.CurveAdd(TrapezoidMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[4, 8, 8, 
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

X1 (Get method)

Gets the value of the first point on the X axis.

```
double  X1()
```

Return Value

The value of the first point on the X axis.

X1 (Set method)

Sets the value of the first point on the X axis.

```
void  X1(
   const double  x1      // value of the first point on the X axis
   )
```

Parameters

x1

[in] The value of the first point on the X axis.

X2 (Get method)

Gets the value of the second point on the X axis.

```
double  X2()
```

Return Value

The value of the second point on the X axis.

X2 (Set method)

Sets the value of the second point on the X axis.

```
void  X2(
   const double  x2      // value of the second point on the X axis
   )
```

Parameters

x2

[in] The value of the second point on the X axis.

X3 (Get method)

Gets the value of the third point on the X axis.

```
double  X3()
```

Return Value

The value of the third point on the X axis.

X3 (Set method)

Sets the value of the third point on the X axis.

```
void  X3(
   const double  x3      // value of the third point on the X axis
   )
```

Parameters

x3

[in] The value of the third point on the X axis.

X4 (Get method)

Gets the value of the fourth point on the X axis.

```
double  X4()
```

Return Value

The value of the fourth point on the X axis.

X4 (Set method)

Sets the value of the fourth point on the X axis.

```
void  X4(
   const double  x4      // value of the fourth point on the X axis
   )
```

Parameters

x4

[in] The value of the fourth point on the X axis.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x        // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

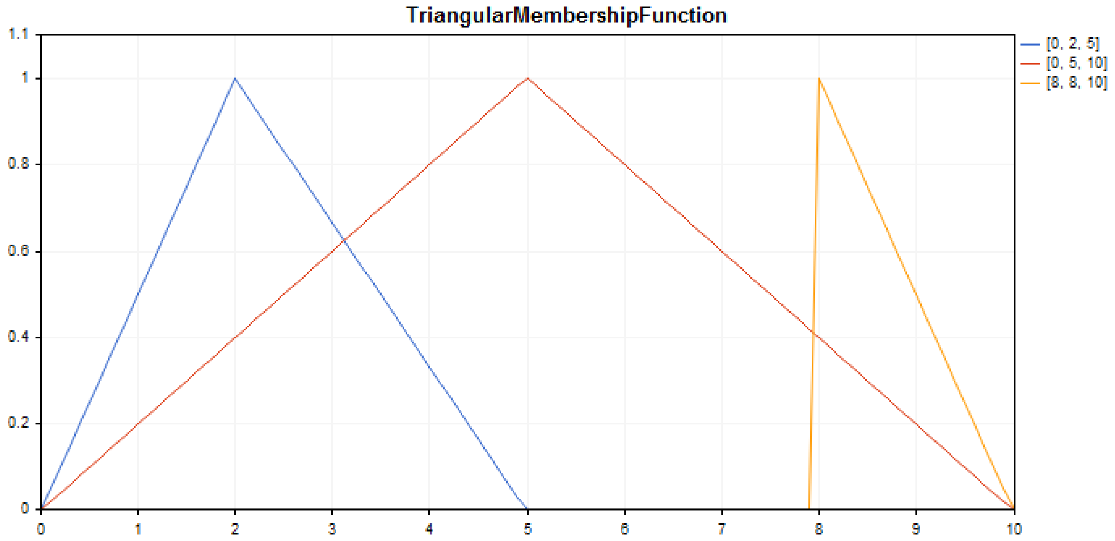
Membership function value.

CTriangularMembershipFunction

Class for implementing a triangle membership function with the X1, X2 and X3 parameters.

Description

The function sets a membership function in the form of a triangle. This is a simple and most frequently applied membership function.



A sample code for plotting a chart is displayed below.

Declaration

```
class CTriangularMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

CObject
  IMembershipFunction
    CTriangularMembershipFunction

Class methods

Class method	Description
X1	Gets the value of the first point on the X axis.
X2	Gets the value of the second point on the X axis.
X3	Gets the value of the third point on the X axis.

Class method	Description
ToNormalMF	Converts a triangle membership function into a Gaussian one.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                 TriangularMembershipFunction.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2000-2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTriangularMembershipFunction func1(0,2,5);
CTriangularMembershipFunction func2(0,5,10);
CTriangularMembershipFunction func3(8,8,10);
//--- Create wrappers for membership functions
double TriangularMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TriangularMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TriangularMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"TriangularMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"TriangularMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("TriangularMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(TriangularMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5]"
   graphic.CurveAdd(TriangularMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 10]
   graphic.CurveAdd(TriangularMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[8, 8, 10]
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

X1 (Get method)

Gets the value of the first point on the X axis.

```
double  X1()
```

Return Value

The value of the first point on the X axis.

X1 (Set method)

Sets the value of the first point on the X axis.

```
void  X1(
   const double  x1       // value of the first point on the X axis
   )
```

Parameters

x1

[in] The value of the first point on the X axis.

X2 (Get method)

Gets the value of the second point on the X axis.

```
double  X2()
```

Return Value

The value of the second point on the X axis.

X2 (Set method)

Sets the value of the second point on the X axis.

```
void  X2(
   const double  x2      // value of the second point on the X axis
   )
```

Parameters

x2

[in] The value of the second point on the X axis.

X3 (Get method)

Gets the value of the third point on the X axis.

```
double  X3()
```

Return Value

The value of the third point on the X axis.

X3 (Set method)

Sets the value of the third point on the X axis.

```
void  X3(
   const double  x3      // value of the third point on the X axis
   )
```

Parameters

x3

[in] The value of the third point on the X axis.

ToNormalMF

Converts a triangle membership function into a Gaussian one.

```
CNormalMembershipFunction*  ToNormalMF()
```

Return Value

The pointer to a Gaussian membership function.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x      // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

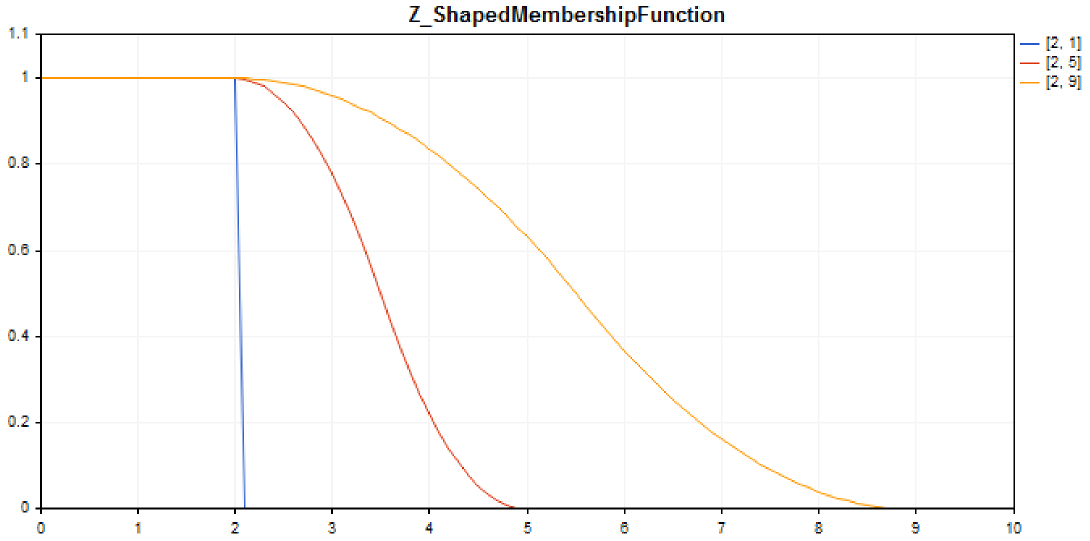
CZ_ShapedMembershipFunction

Class for implementing a z-like membership function with the A and B parameters.

Description

The function sets a z-like two-parameter membership function. This is a non-increasing membership function that takes values from 1 to 0. The function parameters define an interval, within which the function decreases in non-linear trajectory from 1 to 0.

The function represents fuzzy sets of "very low" type. In other words, it sets non-increasing membership functions with saturation.



A sample code for plotting a chart is displayed below.

Declaration

```
class CZ_ShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

COObject

IMembershipFunction

CZ_ShapedMembershipFunction

Class methods

Class method	Description
A	Gets and sets the parameter of the decreasing interval start.

Class method	Description
B	Gets and sets the parameter of the decreasing interval end.
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Example

```
//+------------------------------------------------------------------+
//|                                    Z_ShapedMembershipFunction.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CZ_ShapedMembershipFunction func1(2,1);
CZ_ShapedMembershipFunction func2(2,5);
CZ_ShapedMembershipFunction func3(2,9);
//--- Create wrappers for membership functions
double Z_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double Z_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double Z_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"Z_ShapedMembershipFunction",0,30,30,780,380))
     {
      graphic.Attach(0,"Z_ShapedMembershipFunction");
     }
   graphic.HistoryNameWidth(70);
   graphic.BackgroundMain("Z_ShapedMembershipFunction");
   graphic.BackgroundMainSize(16);
//--- create curve
   graphic.CurveAdd(Z_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
   graphic.CurveAdd(Z_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 5]");
   graphic.CurveAdd(Z_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 9]");
//--- sets the X-axis properties
   graphic.XAxis().AutoScale(false);
   graphic.XAxis().Min(0.0);
   graphic.XAxis().Max(10.0);
   graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
   graphic.YAxis().AutoScale(false);
   graphic.YAxis().Min(0.0);
   graphic.YAxis().Max(1.1);
   graphic.YAxis().DefaultStep(0.2);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

A (Get method)

Gets the parameter of the decreasing interval start.

```
double  A()
```

Return Value

Decreasing interval start parameter.

A (Set method)

Sets the parameter of the decreasing interval start.

```
void  A(
   const double  a       // decreasing interval start parameter
   )
```

Parameters

a

[in] Decreasing interval start parameter.

B (Get method)

Gets the parameter of the decreasing interval end.

```
double  B()
```

Return Value

Decreasing interval end parameter.

B (Set method)

Sets the parameter of the decreasing interval end.

```
void  B(
   const double  b       // decreasing interval end parameter
   )
```

Parameters

b

[in] Decreasing interval end parameter.

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x      // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

IMembershipFunction

Basic class for all membership function classes.

Declaration

```
class CZ_ShapedMembershipFuncion : public IMembershipFunction
```

Title

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

Inheritance hierarchy

CObject

IMembershipFunction

Direct descendants

CCompositeMembershipFunction, CConstantMembershipFunction, CDifferencTwoSigmoidalMembershipFunction, CGeneralizedBellShapedMembershipFunction, CNormalCombinationMembershipFunction, CNormalMembershipFunction, CP_ShapedMembershipFunction, CProductTwoSigmoidalMembershipFunctions, CS_ShapedMembershipFunction, CSigmoidalMembershipFunction, CTrapezoidMembershipFunction, CTriangularMembershipFunction, CZ_ShapedMembershipFunction

Class methods

Class method	Description
GetValue	Calculates the value of the membership function by a specified argument.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

GetValue

Calculates the value of the membership function by a specified argument.

```
double  GetValue(
   const  x       // membership function argument
   )
```

Parameters

x

[in] Membership function argument.

Return Value

Membership function value.

Fuzzy systems rules

A **fuzzy system** (fuzzy logical inference system) is a receipt of conclusion in the form of a fuzzy set corresponding to the current values of the inputs with the use of a set of **fuzzy rules** and fuzzy operations.

The **fuzzy rules** determine the relationship between inputs and outputs of an examined object. Amount of rules in the system is unlimited. The generalized format of fuzzy rules is as follows:

if *rule condition*, **then** *rule conclusion*.

Rule condition describes the current state of the object. *Rule conclusion* describes how the condition affects the object.

Class of rules for fuzzy systems	Description
CMamdaniFuzzyRule	Class for implementing a fuzzy logic rule for the Mamdani algorithm
CSugenoFuzzyRule	Class for implementing a fuzzy logic rule for the Sugeno algorithm
CSingleCondition	The class sets a fuzzy condition expressed by "Fuzzy variable — Fuzzy term" pair.
CConditions	Class defines a set of fuzzy conditions connected to each other by an operator.
CGenericFuzzyRule	Base class for implementing the both types of fuzzy rules.

CMamdaniFuzzyRule

Mamdani-type fuzzy inference — one of the two basic types of fuzzy systems. Output variable values are set using fuzzy terms.

Description

Fuzzy logic rule for the Mamdani algorithm can be described as follows:

$$if(X_1\ is\ a_1) \wedge (X_2\ is\ a_2) \wedge \ldots \wedge (X_n\ is\ a_n)\ then\ (Y\ is\ d)(W)$$

where:

- X = (X1, X2, X3 ... Xn) — vector of input variables;
- Y — output variable;
- a = (a1, a2, a3 ... an) — vector of input variable values;
- d — output variable value;
- W — rule weight.

Declaration

```
class CMamdaniFuzzyRule : public CGenericFuzzyRule
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject
    IParsableRule
        CGenericFuzzyRule
            CMamdaniFuzzyRule

Class methods

Class method	Description
Conclusion	Gets and sets the Mamdani fuzzy rule conclusion
Weight	Gets and sets the Mamdani fuzzy rule weight

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CGenericFuzzyRule

Condition, Condition, CreateCondition, CreateCondition, CreateCondition

Conclusion (Get method)

Gets the Mamdani fuzzy rule conclusion

```
CSingleConditon*  Conclusion()
```

Return Value

Conclusion of a Mamdani fuzzy rule.

Conclusion (Set method)

Sets the Mamdani fuzzy rule conclusion.

```
void  Conclusion(
   CSingleConditon*  value       // conclusion of a Mamdani fuzzy rule
   )
```

Parameters

value

[in] Conclusion of a Mamdani fuzzy rule.

Weight (Get method)

Gets the Mamdani fuzzy rule weight.

```
double  Weight()
```

Return Value

Mamdani fuzzy rule weight.

Weight (Set method)

Sets the Mamdani fuzzy rule weight.

```
void  Weight(
   const double  value       // weight of a Mamdani fuzzy rule
   )
```

Parameters

value

[in] Weight of a Mamdani fuzzy rule.

CSugenoFuzzyRule

Sugeno-type fuzzy inference — one of the two basic types of fuzzy systems. Output variable values are set as a linear combination of input variables.

Description

Unlike the Mamdani rule, an input variable value is set by a linear function from input parameters rather than by a fuzzy term. Fuzzy logic rule for the Sugeno algorithm can be described as follows:

$$if(X_1\ is\ a_1)\wedge(X_2\ is\ a_2)\wedge...\wedge(X_n\ is\ a_n)\ then\ (Y=b_0+b_1\cdot X_1+b_2\cdot X_2+...+b_n\cdot X_n)(W)$$

where:

- X = (X1, X2, X3 ... Xn) — vector of input variables;
- Y — output variable;
- a = (a1, a2, a3 ... an) — vector of input variable values;
- b = (b1, b2, b3 ... bn) — free term ratio in the linear function for an output value
- W — rule weight.

Declaration

```
class CSugenoFuzzyRule : public CGenericFuzzyRule
```

Title

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject
  IParsableRule
    CGenericFuzzyRule
      CSugenoFuzzyRule

Class methods

Class method	Description
Conclusion	Gets and sets the Sugeno fuzzy rule conclusion

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CGenericFuzzyRule

Condition, Condition, CreateCondition, CreateCondition, CreateCondition

Conclusion (Get method)

Gets the Sugeno fuzzy rule conclusion.

```
CSingleConditon*  Conclusion()
```

Return Value

Sugeno fuzzy rule conclusion.

Conclusion (Set method)

Sets the Sugeno fuzzy rule conclusion.

```
void  Conclusion(
   CSingleCondition*  value       // conclusion of a Sugeno fuzzy rule
   )
```

Parameters

value

[in] Conclusion of a Sugeno fuzzy rule.

CSingleCondition

The class sets a fuzzy condition expressed by "Fuzzy variable — Fuzzy term" pair.

Description

According to a fuzzy condition, one variable corresponds to one term. A fuzzy condition can be described by the following expression: *X is a*,

where:

- X is a fuzzy variable;
- a is a fuzzy variable value (fuzzy term).

Declaration

```
    class CSingleCondition : public ICondition
```

Title

```
    #include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject

ICondition

CSingleCondition

Direct descendants

CFuzzyCondition

Class methods

Class method	Description
Not	Gets and sets the flag indicating whether it is necessary to apply negation to this condition.
Term	Gets and sets a fuzzy term for this condition.
Var	Gets and sets a fuzzy variable for this condition.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Not (Get method)

Gets the flag indicating whether it is necessary to apply negation to this condition.

```
bool Not()
```

Return Value

The flag value.

Not (Set method)

Sets the flag indicating whether it is necessary to apply negation to this condition.

```
void  Not(
   bool  not      // flag value
   )
```

Parameters

not

[in] Flag value.

Term (Get method)

Gets a fuzzy term for the given condition.

```
INamedValue* Term()
```

Return Value

A fuzzy term for the given condition.

Term (Set method)

Sets a fuzzy term for the given condition.

```
void  Term(
   INamedValue*&  value      // fuzzy term for the given condition
   )
```

Parameters

value

[in] A fuzzy term for the given condition.

Var (Get method)

Gets a fuzzy variable for the given condition.

```
INamedVariable* Var()
```

Return Value

A fuzzy variable for the given condition.

Var (Set method)

Sets a fuzzy variable for the given condition

```
void  Var(
   INamedVariable*&  value      // a fuzzy variable for the given condition
```

```
    )
```

Parameters

value

[in] fuzzy variable.

CConditions

Class defines a set of fuzzy conditions connected to each other by an operator.

Description

A set of fuzzy conditions connected to each other by an operator may be described as follows:

$$(X_1\ is\ a_1) \wedge (X_2\ is\ a_2) \wedge \ldots \wedge (X_n\ is\ a_n)$$

where:

- X = (X1, X2, X3 ... Xn) — vector of input variables;
- a = (a1, a2, a3 ... an) — vector of input variable values.

In this example, the *and* operator is used. Besides, the *or* operator is available in this class.

Declaration

```
   class CConditions : public ICondition
```

Title

```
   #include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject
    ICondition
        CConditions

Class methods

Class method	Description
ConditionsList	Gets the list of all conditions
Not	Gets and sets the flag indicating whether it is necessary to apply negation to these conditions
Op	Gets and sets a type of the conditions bundle operator.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

ConditionsList

Gets the list of all conditions.

```
CList*  ConditionsList()
```

Return Value

List of all conditions.

Not (Get method)

Gets the flag indicating whether it is necessary to apply negation to these conditions.

```
bool Not()
```

Return Value

The flag value.

Not (Set method)

Sets the flag indicating whether it is necessary to apply negation to these conditions

```
void  Not(
   bool   not       // flag value
   )
```

Parameters

not

[in] Flag value.

Op (Get method)

Gets a type of the conditions bundle operator. The *and* and *or* operators are available.

```
OperatorType Op()
```

Return Value

Type of the conditions bundle operator.

Op (Set method)

Set the conditions bundle operator. The *and* and *or* operators are available.

```
void  Op(
   OperatorType   op      // type of the conditions bundle operator
   )
```

Parameters

op

[in] Type of the conditions bundle operator.

CGenericFuzzyRule

Base class for both types of fuzzy rules.

Declaration

```
   class CGenericFuzzyRule : public IParsableRule
```

Title

```
   #include <Math\Fuzzy\fuzzyrule.mqh>
```

Inheritance hierarchy

CObject
  IParsableRule
    CGenericFuzzyRule

Direct descendants

CMamdaniFuzzyRule, CSugenoFuzzyRule

Class methods

Class method	Description
Conclusion	Gets and sets the fuzzy rule conclusion
Condition	Gets and sets the 'if' condition (set of conditions) for a fuzzy rule
CreateCondition	Creates a condition for a fuzzy rule by specified parameters

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Conclusion (Get method)

Gets the conclusion of a fuzzy rule.

```
CSingleConditon*  Conclusion()
```

Return Value

Conclusion of a fuzzy rule.

Conclusion (Set method)

Sets the fuzzy rule conclusion.

```
virtual void  Conclusion(
   CSingleConditon*  value      // conclusion of a fuzzy rule
   )
```

Parameters

value

[in] Conclusion of a fuzzy rule.

Condition (Get method)

Gets the 'if' condition (set of conditions) for a fuzzy rule.

```
CConditons*  Condition()
```

Return Value

Fuzzy condition (set of conditions).

Condition (Set method)

Sets the 'if' condition (set of conditions) for a fuzzy rule.

```
void  Condition(
   CConditons*  value       // 'if' condition (set of conditions) for a fuzzy rule
   )
```

Parameters

value

[in] Fuzzy condition (set of conditions).

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable*  var,       // fuzzy variable
   CFuzzyTerm*      term,      // fuzzy term
   )
```

Parameters

var

[in] Fuzzy variable.

term

[in] Fuzzy term.

Return Value

Fuzzy rule status.

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable*  var,      // fuzzy variable
   CFuzzyTerm*      term,     // fuzzy term
   bool             not,      // flag indicating whether it is necessary to apply nega
   )
```

Parameters

var

[in] Fuzzy variable.

term

[in] Fuzzy term.

not

[in] Flag indicating whether it is necessary to apply negation to a condition.

Return Value

Fuzzy rule status.

CreateCondition

Creates a condition for a fuzzy rule by specified parameters.

```
CFuzzyCondition*  CreateCondition(
   CFuzzyVariable* var,      // fuzzy variable
   CFuzzyTerm*     term,     // fuzzy term
   bool            not,      // flag indicating whether it is necessary to apply negat
   HedgeType       hedge     // condition bundle type
   )
```

Parameters

var

[in] Fuzzy variable.

term

[in] Fuzzy term.

not

[in] Flag indicating whether it is necessary to apply negation to a condition.

hedge

[in] Condition bundle type.

Return Value

Fuzzy rule status.

Fuzzy systems variables

Fuzzy (linguistic) variables **are applied in fuzzy systems**. These are the variables whose values are words or word combinations in a natural or artificial language.

Linguistic variables comprise fuzzy sets. The nature and number of fuzzy variables change for each certain task when defining fuzzy sets.

Class	Description
CFuzzyVariable	Class for creating general fuzzy variables.
CSugenoVariable	Class for creating fuzzy Sugeno-type variables.

CFuzzyVariable

Class for creating general fuzzy variables.

Description

Here, a fuzzy variable is created with the following parameters:

- maximum variable value;
- minimum variable value;
- fuzzy variable name;
- term set (set of all possible values, which a linguistic variable is capable of receiving).

Declaration

```
   class CFuzzyVariable : public CNamedVariableImpl
```

Title

```
   #include <Math\Fuzzy\fuzzyvariable.mqh>
```

Inheritance hierarchy

CObject
  INamedValue
    INamedVariable
      CNamedVariableImpl
        CFuzzyVariable

Class methods

Class method	Description
AddTerm	Adds a single fuzzy term to a fuzzy variable.
GetTermByName	Gets a fuzzy term by a specified name.
Max	Gets and sets a maximum value for a fuzzy variable.
Min	Gets and sets a minimum value for a fuzzy variable.
Terms	Gets and sets a list of fuzzy terms for the given fuzzy variable.
Values	Gets and sets a list of fuzzy terms for the given fuzzy variable.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CNamedVariableImpl

Name, Name

AddTerm

Adds a single fuzzy term to a fuzzy variable.

```
void  AddTerm(
   CFuzzyTerm*&   term      // fuzzy term
   )
```

Parameters

term

[in] Fuzzy term.

GetTermByName

Gets a fuzzy term by a specified name.

```
CFuzzyTerm*  GetTermByName(
   const string   name      // fuzzy term name
   )
```

Parameters

name

[in] Fuzzy term name.

Return Value

Fuzzy term with a specified name.

Max (Get method)

Gets the maximum value for a fuzzy variable.

```
double  Max()
```

Return Value

Maximum value for a fuzzy variable.

Max (Set method)

Sets the maximum value for a fuzzy variable.

```
void  Max(
   const double   max      // maximum value for a fuzzy variable
   )
```

Parameters

max

[in] Maximum value for a fuzzy variable.

Min (Get method)

Gets the minimum value for a fuzzy variable.

```
double  Min()
```

Return Value

Minimum value for a fuzzy variable.

Max (Set method)

Sets the minimum value for a fuzzy variable.

```
void  Min(
   const double  min       // minimum value for a fuzzy variable
   )
```

Parameters

min

[in] Minimum value for a fuzzy variable.

Terms (Get method)

Gets a list of fuzzy terms for the given fuzzy variable.

```
CList*  Terms()
```

Return Value

List of fuzzy terms for the given fuzzy variable.

Terms (Set method)

Sets a list of fuzzy terms for the given fuzzy variable.

```
void  Terms(
   CList*&  terms      // list of fuzzy terms for the given variable
   )
```

Parameters

terms

[in] list of fuzzy terms for the given fuzzy variable.

Values

Gets a list of fuzzy terms for the given fuzzy variable.

```
CList*  Values()
```

Return Value

List of fuzzy terms for the given variable.

CSugenoVariable

Class for creating fuzzy Sugeno-type variables.

Description

Fuzzy Sugeno-type variable is different from the general linguistic variable since it is not set by a term set but by a set of linear functions.

Declaration

```
class CSugenoVariable : public CNamedVariableImpl
```

Title

```
#include <Math\Fuzzy\sugenovariable.mqh>
```

Inheritance hierarchy

CObject
  INamedValue
    INamedVariable
      CNamedVariableImpl
        CSugenoVariable

Class methods

Class method	Description
Functions	Gets the list of linear functions of the fuzzy Sugeno variable.
GetFuncByName	Gets the linear function by a specified name.
Values	Gets the list of linear functions of the fuzzy Sugeno variable.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CNamedVariableImpl

Name, Name

Functions

Gets the list of linear functions of the fuzzy Sugeno variable.

```
CList*  Functions()
```

Return Value

List of linear functions.

GetFuncByName

Gets the linear function by a specified name.

```
ISugenoFunction*  GetFuncByName(
   const string  name      // linear function name
   )
```

Parameters

name

[in] Linear function name.

Return Value

Linear function with a specified name.

Values

Gets the list of linear functions of the fuzzy Sugeno variable.

```
CList*  Values()
```

Return Value

List of linear functions of the fuzzy Sugeno variable.

CFuzzyTerm (fuzzy terms)

Class for implementing fuzzy terms.

Description

A **term** is any element of a term set. A term is defined by two components:

- fuzzy term name;
- membership function.

Declaration

```
class CFuzzyTerm : public CNamedValueImpl
```

Title

```
#include <Math\Fuzzy\fuzzyterm.mqh>
```

Inheritance hierarchy

CObject
    INamedValue
        CNamedValueImpl
            CFuzzyTerm

Class methods

Class method	Description
MembershipFunction	Gets a membership function for the fuzzy term.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CNamedValueImpl

Name, Name

MembershipFunction

Gets a membership function for the fuzzy term.

```
IMembershipFunction*  MembershipFunction()
```

Return Value

Membership function

Fuzzy systems

Fuzzy system (or *fuzzy model*) is a mathematical model whose calculation is based on fuzzy logic. Construction of such models is applicable when the subject of study has a weak formalization and its exact mathematical description is too complex or unknown.

The progress of a model construction can be divided into three main stages:

1. Definition of input and output characteristics of a model.
2. Building a knowledge base.
3. Selecting one of the methods of fuzzy inference (Mamdani and Sugeno).

The first stage directly effects the consequent two and determines the future operation of the model.

A knowledge base (*rule base*) is a set of fuzzy rules of "if, then" type that define the relationship between inputs and outputs of the examined object.

Rule condition describes the current state of the object, and **rule conclusion** — how this condition affects the object.

There can be two types of terms and conclusions for each rule:

1. simple (link to Csinglcond) — includes one fuzzy variable;
2. complex (link to Cconditions) — includes several fuzzy variables.

Each rule in the system has its **weight** — importance of a rule in the model. Weighting factors are assigned to a rule within range [0, 1].

Depending on the created knowledge base, the system of fuzzy inference is determined for a model. **Fuzzy logical inference** is a receipt of conclusion in form of a fuzzy set corresponding to the current value of the inputs with use of knowledge base and fuzzy operations. The two main types of fuzzy inference are Mamdani and Sugeno.

Mamdani system

Output variable values in the Mamdani system are set using fuzzy terms.

Description

Fuzzy logic rule for the Mamdani algorithm can be described as follows:

$$if(X_1\ is\ a_1) \wedge (X_2\ is\ a_2) \wedge \ldots \wedge (X_n\ is\ a_n)\ then\ (Y\ is\ d)(W)$$

where:

- X = (X1, X2, X3 ... Xn) — vector of input variables;
- Y — output variable;
- a = (a1, a2, a3 ... an) — vector of input variable values;
- d — output variable value;
- W — rule weight.

Class methods

Class method	Description
AggregationMethod	Sets the type of conditions aggregation
Calculate	Calculates a fuzzy inference for the system
DefuzzificationMethod	Sets defuzzification method type
EmptyRule	Creates an empty fuzzy Mamdani rule based on the current system
ImplicationMethod	Sets a type of the system implication operator
Output	Gets the list of fuzzy Mamdani output variables.
OutputByName	Gets a fuzzy Mamdani output variable by a specified name.
ParseRule	Creates a fuzzy Mamdani rule based on a specified line.
Rules	Returns the list of fuzzy Mamdani rules.

Methods inherited from class CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

AggregationMethod

Sets the type of conditions aggregation method.

```
void  AggregationMethod(
   AggregationMethod  value      // aggregation method type
```

```
    )
```

Parameters

value

[in] Type of conditions aggregation method.

Calculate

Calculates a fuzzy inference for the system

```
CList*  Calculate(
   CList*  inputValues      // input data
   )
```

Parameters

inputValues

[in] Input data for calculation.

Return Value

Calculation result.

DefuzzificationMethod

Sets defuzzification method type.

```
void  DefuzzificationMethod(
   DefuzzificationMethod  value      // defuzzification method type.
   )
```

Parameters

value

[in] Defuzzification method type.

EmptyRule

Creates an empty fuzzy Mamdani rule based on the current system

```
CMamdaniFuzzyRule*  EmptyRule()
```

Return Value

Fuzzy Mamdani rule.

ImplicationMethod

Sets a type of the conditions implication operator.

```
void  ImplicationMethod(
   ImplicationMethod  value      // implication operator type
   )
```

Parameters

value

[in] Type of the conditions implication operator.

Output

Gets the list of fuzzy Mamdani output variables.

```
CList*  Output()
```

Return Value

List of fuzzy variables.

OutputByName

Gets a fuzzy Mamdani output variable by a specified name.

```
CFuzzyVariable*  OutputByName(
   const string  name      // fuzzy variable name
   )
```

Parameters

name

[in] Fuzzy variable name.

Return Value

Fuzzy Mamdani variable with a specified name.

ParseRule

Creates a fuzzy Mamdani rule based on a specified line.

```
CMamdaniFuzzyRule*  ParseRule(
   const string  rule      // string representation of a fuzzy rule
   )
```

Parameters

rule

[in] String representation of a fuzzy Mamdani rule.

Return Value

Fuzzy Mamdani rule.

Rules

Returns the list of fuzzy Mamdani rules.

```
CList*  Rules()
```

Return Value

List of fuzzy Mamdani rules.

Sugeno system

Sugeno fuzzy logic system is one of the two basic types of fuzzy systems. Output variable values are set as a linear combination of input variables.

Description

Unlike the Mamdani rule, an input variable value is set by a linear function from entries rather than by a fuzzy term. Fuzzy logic rule for the Sugeno algorithm can be described as follows:

$$if(X_1\ is\ a_1)\wedge(X_2\ is\ a_2)\wedge...\wedge(X_n\ is\ a_n)\ then\ (Y=b_0+b_1\cdot X_1+b_2\cdot X_2+...+b_n\cdot X_n)(W)$$

where:

- X = (X1, X2, X3 ... Xn) — vector of input variables;
- Y — output variable;
- a = (a1, a2, a3 ... an) — vector of input variable values;
- b = (b1, b2, b3 ... bn) — free term ratio in the linear function for an output value
- W — rule weight.

Class methods

Class method	Description
Calculate	Calculates a fuzzy inference for the system
CreateSugenoFunction	Creates a linear Sugeno function for the system.
EmptyRule	Creates an empty fuzzy Sugeno rule based on the current system.
Output	Gets the list of fuzzy Sugeno output variables.
OutputByName	Gets a fuzzy Sugeno output variable by a specified name.
ParseRule	Creates a fuzzy Sugeno rule based on a specified line.
Rules	Returns the list of fuzzy rules.

Methods inherited from class CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

Calculate

Calculates a fuzzy inference for the system

```
CList*  Calculate(
   CList*&  inputValues        // input data
   )
```

Parameters

inputValues

[in] Input data for calculation.

Return Value

Calculation result.

CreateSugenoFunction

Creates a linear Sugeno function for the system.

```
CLinearSugenoFunction*  CreateSugenoFunction(
   const string  name,           // function name
   const double& coeffs[]        // function ratios
   )
```

Parameters

name

[in] Function name.

coeffs[]

[in] Function ratios.

Return Value

Linear Sugeno function.

Note

The size of the ratio array may be equal to a number of inputs or exceed that number by one. In the first case, the free term of the Sugeno linear function is equal to zero, while in the second case, it is equal to the last ratio.

CreateSugenoFunction

Creates a linear Sugeno function for the system.

```
CLinearSugenoFunction*  CreateSugenoFunction(
   const string  name,           // function name
   CList*&       coeffs,         // list of pairs fuzzy variable - its ratio
   const double  constValue      // function free term ratio
   )
```

Parameters

name

[in] Function name.

coeffs[]

[in] Function ratios.

Return Value

Linear Sugeno function.

EmptyRule

Creates an empty fuzzy Sugeno rule based on the current system.

```
CSugenoFuzzyRule*  EmptyRule()
```

Return Value

Fuzzy Sugeno rule.

Output

Gets the list of fuzzy Sugeno output variables.

```
CList*  Output()
```

Return Value

List of fuzzy variables.

OutputByName

Gets a fuzzy Sugeno output variable by a specified name.

```
CSugenoVariable*  OutputByName(
   const string  name      // fuzzy variable name
   )
```

Parameters

name

Fuzzy variable name.

Return Value

Fuzzy Sugeno variable with a specified name.

ParseRule

Creates a fuzzy Sugeno rule based on a specified line.

```
CSugenoFuzzyRule*  ParseRule(
   const string  rule      // string representation of a fuzzy Sugeno rule
   )
```

Parameters

rule

[in] String representation of a fuzzy Sugeno rule.

Return Value

Fuzzy Sugeno rule.

Rules

Returns the list of fuzzy rules.

```
CList*  Rules()
```

Return Value

List of fuzzy rules.

Class for working with OpenCL programs

The COpenCL class is a wrapper to facilitate working with the OpenCL functions. In some cases, use of the GPU allows to substantially increase the speed of computations.

Examples of class use for calculations based on float and double values can be found in the corresponding subdirectories of the MQL5\Scripts\Examples\OpenCL\ folder. The source codes of the OpenCL programs are located in MQL5\Scripts\Examples\OpenCL\Double\Kernels and MQL5\Scripts\Examples\OpenCL\Float\Kernels subdirectories.

- MatrixMult.mq5 - example of matrix multiplication using global and local memory
- BitonicSort.mq5 - example of parallel sorting of array elements in GPU
- FFT.mq5 - example of fast Fourier transform calculation
- Wavelet.mq5 - example of wavelet transform of data using the Morlet wavelet.

It is recommended to write the source code for OpenCL in separate CL files, which can later be included in the MQL5 program using the resource variables.

Declaration

```
   class COpenCL
```

Title

```
   #include <OpenCL\OpenCL.mqh>
```

Class methods

Name	Description
BufferCreate	Creates an OpenCL buffer at the specified index
BufferFree	Deletes buffer at the specified index
BufferFromArray	Creates a buffer at the specified index from an array of values
BufferRead	Reads an OpenCL buffer at the specified index into an array
BufferWrite	Writes an array of values into buffer at the specified index
Execute	Executes the OpenCL kernel with the specified index
GetContext	Returns handle of the OpenCL context
GetKernel	Returns handle of the kernel object at the specified index
GetKernelName	Returns name of the kernel object at the specified index
GetProgram	Returns handle of the OpenCL program
Initialize	Initializes the OpenCL program
KernelCreate	Creates an entry point into the OpenCL program at the specified index
KernelFree	Removes an OpenCL start function at the specified index

Name	Description
SetArgument	Sets a parameter for the OpenCL function at the specified index
SetArgumentBuffer	Sets an OpenCL buffer as a parameter of the OpenCL function at the specified index
SetArgumentLocalMemory	Sets a parameter in local memory for the OpenCL function at the specified index
SetBuffersCount	Sets the number of buffers
SetKernelsCount	Sets the number of kernel objects
Shutdown	Unloads the OpenCL program
SupportDouble	Checks if floating point data types are supported on the device

BufferCreate

Creates an OpenCL buffer at the specified index.

```
bool  BufferCreate(
   const int    buffer_index,       // buffer index
   const uint   size_in_bytes,      // buffer size in bytes
   const uint   flags               // combination of flags that define the buffer prope
   );
```

Parameters

buffer_index

[in] Index buffer.

size_in_bytes

[in] Buffer size in bytes.

flags

[in] Buffer properties that are set using a combination of flags.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferFree

Deletes buffer at the specified index.

```
bool  BufferFree(
   const int  buffer_index       // buffer index
   );
```

Parameters

buffer_index

[in] Index buffer.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferFromArray

Creates a buffer at the specified index from an array of values.

```
template<typename T>
bool  BufferFromArray(
   const int    buffer_index,          // buffer index
   T            &data[],               // array of values
   const uint   data_array_offset,     // offset in the array of values, in bytes
   const uint   data_array_count,      // number of values from the array to write
   const uint   flags                  // combination of flags that define the buffer 
   );
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] An array of values to be written into the OpenCL buffer.

data_array_offset

[in] Offset in the array of values in bytes, from which writing of values begins.

data_array_count

[in] The number of values to be written.

flags

[in] Buffer properties that are set using a combination of flags.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferRead

Reads an OpenCL buffer at the specified index into an array.

```
template<typename T>
bool  BufferRead(
   const int    buffer_index,          // buffer index
   T            &data[],               // array of values
   const uint   cl_buffer_offset,      // offset in the OpenCL buffer, in bytes
   const uint   data_array_offset,     // shift in the array elements
   const uint   data_array_count       // number of values from the buffer to read
   );
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] Array to obtain the values of the OpenCL buffer.

cl_buffer_offset

[in] Offset in the OpenCL buffer in bytes, from which to start reading values.

data_array_offset

[in] Index of the first element of the array to write values of the OpenCL buffer.

data_array_count

[in] The number of values to be read.

Return Value

In case of successful execution, returns true, otherwise - false.

BufferWrite

Writes an array of values into buffer at the specified index.

```
template<typename T>
bool  BufferWrite(
   const int    buffer_index,          // buffer index
   T            &data[],               // array of values
   const uint   cl_buffer_offset,      // offset in the OpenCL buffer, in bytes
   const uint   data_array_offset,     // shift in the array elements
   const uint   data_array_count       // number of values from the array to write
   );
```

Parameters

buffer_index

[in] Index buffer.

&data[]

[in] An array of values to be written into the OpenCL buffer.

cl_buffer_offset

[in] Offset in the OpenCL buffer in bytes, from which to start writing values.

data_array_offset

[in] Index of the first element of the array, starting from which the array values are written to the OpenCL buffer.

data_array_count

[in] The number of values to be written.

Return Value

In case of successful execution, returns true, otherwise - false.

Execute

Executes the OpenCL program at the specified index.

```
bool  Execute(
   const int    kernel_index,            // index of the kernel
   const int    work_dim,                // dimension of the tasks space
   const uint   &work_offset[],          // initial offset in the tasks space
   const uint   &work_size[]             // total number of tasks
   );
```

Executes the OpenCL kernel with the specified index and number of tasks in the local group.

```
bool  Execute(
   const int    kernel_index,            // index of the kernel
   const int    work_dim,                // dimension of the tasks space
   const uint   &work_offset[],          // initial offset in the tasks space
   const uint   &work_size[],            // total number of tasks
   const uint   &local_work_size[]       // number of tasks in the local group
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

work_dim

[in] Dimension of the tasks space.

&work_offset[]

[in][out] Initial offset in the tasks space. Passed by reference.

&work_size[]

[in][out] The size of the tasks subset. Passed by reference.

&local_work_size[]

[in][out] The size of the local tasks subset in the group. Passed by reference.

Return Value

In case of successful execution, returns true, otherwise - false.

GetContext

Returns handle of the OpenCL context.

```
int  GetContext();
```

Return Value

Handle of the OpenCL context.

GetKernel

Returns handle of the kernel object at the specified index.

```
int  GetKernel(
   const int  kernel_index      // index of the kernel
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

Handle of the kernel object.

GetKernelName

Returns name of the kernel object at the specified index.

```
string  GetKernelName(
   const int  kernel_index      // index of the kernel
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

Name of the kernel object.

GetProgram

Returns handle of the OpenCL program.

```
int  GetProgram();
```

Return Value

Handle of the OpenCL program.

Initialize

Initializes the OpenCL program.

```
bool  Initialize(
   const string  program,            // handle of the OpenCL program
   const bool    show_log=true       // keep a log
   );
```

Parameters

program

[in] Handle of the OpenCL program.

show_log=true

[in] Enable logging messages.

Return Value

Returns true, if the initialization succeeded. Otherwise, it returns false.

KernelCreate

Creates an entry point into the OpenCL program at the specified index.

```
bool  KernelCreate(
   const int      kernel_index,      // index of the kernel
   const string  kernel_name         // name of the kernel
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

kernel_name

[in] Name of the kernel object.

Return Value

In case of successful execution, returns true, otherwise - false.

KernelFree

Removes an OpenCL start function at the specified index.

```
bool  KernelFree(
   const int  kernel_index      // index of the kernel
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgument

Sets a parameter for the OpenCL function at the specified index.

```
template<typename T>
bool  SetArgument(
   const int  kernel_index,      // index of the kernel
   const int  arg_index,         // index of the function argument
   T          value              // source code
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

value

[in] The value of the function argument.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgumentBuffer

Sets an OpenCL buffer as a parameter of the OpenCL function at the specified index.

```
bool  SetArgumentBuffer(
   const int  kernel_index,      // index of the kernel
   const int  arg_index,         // index of the function argument
   const int  buffer_index       // buffer index
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

buffer_index

[in] Index buffer.

Return Value

In case of successful execution, returns true, otherwise - false.

SetArgumentLocalMemory

Sets a parameter in local memory for the OpenCL function at the specified index.

```
bool  SetArgumentLocalMemory(
   const int  kernel_index,          // index of the kernel
   const int  arg_index,             // index of the function argument
   const int  local_memory_size      // size of the local memory
   );
```

Parameters

kernel_index

[in] Index of the kernel object.

arg_index

[in] Index of the function argument.

local_memory_size

[in] Size of the local memory.

Return Value

In case of successful execution, returns true, otherwise - false.

SetBuffersCount

Sets the number of buffers.

```
bool  SetBuffersCount(
   const int  total_buffers      // number of buffers
   );
```

Parameters

total_buffers

[in] The total number of buffers.

Return Value

In case of successful execution, returns true, otherwise - false.

SetKernelsCount

Sets the number of kernel objects.

```
bool  SetKernelsCount(
   const int  total_kernels       // number of kernels
   );
```

Parameters

total_kernels

[in] The total number of kernels.

Return Value

In case of successful execution, returns true, otherwise - false.

Shutdown

Unloads the OpenCL program.

```
void  Shutdown();
```

Return Value

No return value.

SupportDouble

Checks if floating point data types are supported on the device.

```
bool  SupportDouble();
```

Return Value

Returns true, if the device supports floating point data types.

Basic Class CObject

Class CObject is the base class for constructing a MQL5 Standard Library.

Description

Class CObject allows all its descendants to be part of a linked list. Also, a number of virtual methods for further implementation in descendant classes are identified.

Declaration

```
class CObject
```

Title

```
#include <Object.mqh>
```

Inheritance hierarchy

CObject

Direct descendants

CAccountInfo, CArray, CChart, CChartObject, CCurve, CDealInfo, CDictionary_Obj_Double, CDictionary_Obj_Obj, CDictionary_String_Obj, CExpertBase, CFile, CHistoryOrderInfo, CList, COrderInfo, CPositionInfo, CString, CSymbolInfo, CTerminalInfo, CTrade, CTreeNode, CWnd, ICondition, IExpression, IMembershipFunction, INamedValue, IParsableRule

Class Methods by Groups

Attributes	
Prev	Gets the value of the previous item
Prev	Sets the value of the previous item
Next	Gets the value of the subsequent element
Next	Sets the next element
Compare methods	
virtual Compare	Returns the result of comparison with another object
Input/output	
virtual Save	Writes object to a file
virtual Load	Reads the object from the file
virtual Type	Returns the type of object

Prev

Gets a pointer to the previous element in the list.

```
CObject*  Prev()
```

Return Value

Pointer to the previous element in the list. If an item is listed first, then return NULL.

Example:

```
//--- example for CObject::Prev()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use prev object
   CObject *object=object_second.Prev();
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Prev

Sets the pointer to the previous element in the list.

```
void  Prev(
   CObject*  object       // Pointer to the previous element in the list
   )
```

Parameters

object

[in] New value of the pointer to the previous element in the list.

Example:

```
//--- example for CObject::Prev(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use objects
   //--- ...
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Next

Gets a pointer to the next element in the list.

```
CObject*  Next()
```

Return Value

Pointer to the next element in the list. If this is the last element in the list, return NULL.

Example:

```
//--- example for CObject::Next()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use next object
   CObject *object=object_first.Next();
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Next

Sets the pointer to the next element in the list.

```
void  Next(
   CObject*   object       // Pointer to the next element in the list
   )
```

Parameters

object

[in] New value of the pointer to the next element in the list.

Example:

```
//--- example for CObject::Next(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use objects
   //--- ...
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Compare

Compares the data on a list element with data on another list element.

```
virtual int  Compare(
   const CObject*  node,      // element
   const int       mode=0     // variant
   ) const
```

Parameters

node

[in] Pointer to a list element to compare

mode=0

[in] Comparison variant

Return Value

0 - in case the list elements are equal, -1 - if the list element is less than the element used for comparison (node), 1 - if the list element is greater than the element used for comparison (node).

Note

Compare() method in CObject class always returns 0 and does not perform any action. If you want to compare data in derived class, the Compare(...) method should be implemented. The 'mode' parameter should be used when implementing multivariate comparison.

Example:

```
//--- example for CObject::Compare(...)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- compare objects
   int result=object_first.Compare(object_second);
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Save

Saves list element data in a file.

```
virtual bool  Save(
   int   file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle of the binary file opened earlier using the FileOpen () function

Return Value

true - successfully completed, false - error.

Note

Save(int) method in CObject class always returns 'true' and does not perform any action. If you want to save the data of a derived class in a file, the Save(int) method should be implemented.

Example:

```
//--- example for CObject::Save(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set objects data
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete object;
  }
```

Load

Loads list element data from a file.

```
virtual bool  Load(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file opened earlier using the FileOpen() function.

Return Value

true - successfully completed, false - error.

Note

Load(int) method in the CObject class always returns 'true' and does not perform any action. If you want to load the data of a derived class from a file, the Load(int) method should be implemented.

Example:

```
//--- example for CObject::Load(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use object
   //--- . . .
   delete object;
  }
```

Type

Gets the type identifier.

```
virtual int  Type() const
```

Return Value

Type identifier (for CObject - 0).

Example:

```
//--- example for CObject::Type()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object=new CObject;
   //---
   object=new CObject;
   if(object ==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get objects type
   int type=object.Type();
   //--- delete object
   delete object;
  }
```

Data Structures

This section contains the technical details on working with various data structures (arrays, linked lists, etc.) and description of the relevant components of the MQL5 Standard Library.

Using classes of data structures will save time when creating custom data stores of various formats (including composite data structures).

MQL5 Standard Library (in terms of data sets) is placed in the working directory of the terminal in the Include\Arrays folder.

Data Arrays

Use of classes of dynamic data arrays will save time when creating a custom data stores of various formats (including multidimensional arrays).

MQL5 Standard Library (in terms of arrays of data) is located in the working directory of the terminal in the Include\Arrays folder.

Class	Description
CArray	Base class of dynamic data array
CArrayChar	Dynamic array of char or uchar variables
CArrayShort	Dynamic array of short or ushort variables
CArrayInt	Dynamic array of int or uint variables
CArrayLong	Dynamic array of long or ulong variables
CArrayFloat	Dynamic array of float variables
CArrayDouble	Dynamic array of double variables
CArrayString	Dynamic array of string variables
CArrayObj	Dynamic array of CObject pointers
CList	Provides the ability to work with a list of instances of CObject class and its descendants
CTreeNode	Provides the ability to work with nodes of the CTree binary tree
CTree	Provides the ability to work with the binary tree of the CTreeNode class instances and its descendants

CArray

CArray class is the base class of a dynamic array of variables.

Description

Class CArray is intended to operate on dynamic arrays of variables: memory allocation, sorting, and working with files.

Declaration

```
class CArray : public CObject
```

Title

```
#include <Arrays\Array.mqh>
```

Inheritance hierarchy

CObject

CArray

Direct descendants

CArrayChar, CArrayDouble, CArrayFloat, CArrayInt, CArrayLong, CArrayObj, CArrayShort, CArrayString

Class Methods by Groups

Attributes	
Step	Gets the increment size of the array
Step	Sets the increment size of the array
Total	Gets the number of elements in the array
Available	Gets the number of free elements of the array that are available without additional memory allocation
Max	Gets the maximum possible size of the array without memory reallocation
IsSorted	Gets the flag of array being sorted using specified sorting mode
SortMode	Gets the sorting mode for an array
Clear methods	
Clear	Deletes all of the array elements without memory release
Sort methods	
Sort	Sorts an array to the specified option
Input/output	

Attributes	
virtual Save	Saves data array in a file
virtual Load	Loads data array from a file

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Step

Gets the increment size of the array.

```
int  Step() const
```

Return Value

Increment size of the array.

Example:

```
//--- example for CArray::Step()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get resize step
   int step=array.Step();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Step

Sets the increment size of the array.

```
bool  Step(
   int  step       // step
   )
```

Parameters

step

[in] The new value of the increment size of the array.

Return Value

true - successful, false - there was an attempt to establish a step less than or equal to zero.

Example:

```
//--- example for CArray::Step(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set resize step
   bool result=array.Step(1024);
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Total

Gets the number of elements in the array.

```
int  Total() const;
```

Return Value

Number of elements in the array.

Example:

```
//--- example for CArray::Total()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check total
   int total=array.Total();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Available

Gets the number of free elements of the array that are available without additional memory allocation.

```
int  Available() const
```

Return Value

Number of free elements of the array available without additional memory allocation.

Example:

```
//--- example for CArray::Available()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check available
   int available=array.Available();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Max

Gets the maximum possible size of the array without memory reallocation.

```
int  Max() const
```

Return Value

The maximum possible size of the array without memory reallocation.

Example:

```
//--- example for CArray::Max()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check maximum size
   int max=array.Max();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

IsSorted

Gets the flag of array being sorted using the specified sorting mode.

```
bool  IsSorted(
   int   mode=0        // sorting mode
   ) const
```

Parameters

mode=0

[in] Tested sorting mode.

Return Value

Flag of the sorted list. If the list is sorted using the specified mode - true, otherwise - false.

Note

The sort flag cannot be changed directly. It is set by the Sort() method and reset by any add/insert method except for the InserSort(...).

Example:

```
//--- example for CArray::IsSorted()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sorted
   if(array.IsSorted())
     {
      //--- use methods for sorted array
      //--- ...
     }
   //--- delete array
   delete array;
  }
```

SortMode

Gets the sorting mode for an array.

```
int  SortMode() const;
```

Return Value

Sorting mode.

Example:

```
//--- example for CArray::SortMode()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sort mode
   int sort_mode=array.SortMode();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Clear

Deletes all of the array elements without memory release.

```
void  Clear()
```

Return Value

None.

Example:

```
//--- example for CArray::Clear()
#include <Arrays\Array.mqh>
//---
void OnStart()
 {
 CArray *array=new CArray;
 //---
 if(array==NULL)
 {
 printf("Object create error");
 return;
 }
 //--- use array
 //--- ...
 //--- clear array
 array.Clear();
 //--- delete array
 delete array;
 }
```

Sort

Sorts an array using the specified option.

```
void  Sort(
   int   mode=0       // sorting mode
   )
```

Parameters

mode=0

[in] Mode of array sorting.

Return Value

No.

Note

Sorting an array is always ascending. For arrays of primitive data types (CArrayChar, CArrayShort, etc.), the 'mode' parameter is not used. For the CArrayObj array, multivariate sorting should be implemented in the Sort(int) method of derived classes.

Example:

```
//--- example for CArray::Sort(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- sorting by mode 0
   array.Sort(0);
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of a binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArray::Save(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int      file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of a binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArray::Load(...)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int      file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

CArrayChar

CArrayChar is a class of dynamic array of char or uchar variables.

Description

CArrayChar class provides the ability to work with a dynamic array of char or uchar variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayChar : public CArray
```

Title

```
#include <Arrays\ArrayChar.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayChar

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Modify methods	
Update	Changes the element at the specified array position

Memory control	
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array methods	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the array type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size        // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is increased by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayChar::Reserve(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayChar::Resize(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayChar::Shutdown()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   char   element       // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayChar::Add(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const char&  src[]      // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayChar::AddArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayChar*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayChar class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayChar::AddArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   char  element,      // element to insert
   int   pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayChar::Insert(char,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const char&  src[],      // source array
   int            pos       // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayChar::InsertArray(const char &[],int)
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayChar*  src,      // pointer to the source
   int           pos      // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayChar::InsertArray(const CArrayChar*,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
//--- delete source array
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const char&   src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayChar::AssignArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayChar*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayChar::AssignArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src   =new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays are identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int    pos,          // position
   char   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayChar::Update(int,char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,'A'))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int   pos,        // position
   int   shift       // value
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayChar::Shift(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos       // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayChar::Delete(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of the last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayChar::DeleteRange(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
char  At(
   int  pos       // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, CHAR_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, CHAR_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayChar::At(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      char result=array.At(i);
      if(result==CHAR_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const char&  src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayChar::CompareArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const CArrayChar*  src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayChar class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayChar::CompareArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   char   element       // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayChar::InsertSort(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort('A'))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   char   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::Search(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search('A')!=-1) printf("Element found");
   else                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   char   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchGreat(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat('A')!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   char   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLess(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess('A')!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   char  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchGreatOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual('A')!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   char   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLessOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual('A')!=-1) printf("Element found");
   else                                 printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   char   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchFirst(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst('A')!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   char   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayChar::SearchLast(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast('A')!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   char   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayChar::SearchLinear(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear('A')!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayChar::Save(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayChar::Load(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = '%c'",i,array.At(i));
     }
   //--- delete array
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayChar - 77).

Example:

```
//--- example for CArrayChar::Type()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayShort

CArrayShort is a class of dynamic array of short or ushort variables.

Description

Class CArrayShort provides the ability to work with a dynamic array of short or ushort variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
   class CArrayShort : public CArray
```

Title

```
   #include <Arrays\ArrayShort.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayShort

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	

Memory control	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayShort::Reserve(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayShort::Resize(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayShort::Shutdown()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   short   element      // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayShort::Add(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const short&   src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayShort::AddArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayShort*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayShort class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayShort::AddArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
  //--- use array
  //--- . . .
  //--- delete array
  delete array;
 }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   short  element,      // element to insert
   int    pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayShort::Insert(short,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const short&  src[],      // source array
   int            pos        // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayShort::InsertArray(const short &[],int)
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayShort*  src,      // pointer to the source
   int            pos      // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements to insert.

pos

[in] Position in the array to insert.

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayShort::InsertArray(const CArrayShort*,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
//--- delete source array
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const short&  src[]      // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayShort::AssignArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayShort*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayShort::AssignArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src  =new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int     pos,          // position
   short   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayShort::Update(int,short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // positions
   int  shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayShort::Shift(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos       // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayShort::Delete(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of the last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayShort::DeleteRange(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
short  At(
   int  pos       // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, SHORT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, SHORT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayShort::At(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      short result=array.At(i);
      if(result==SHORT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const short&  src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayShort::CompareArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const CArrayShort*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayShort class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayShort::CompareArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   short   element      // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayShort::InsertSort(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::Search(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100)!=-1) printf("Element found");
   else                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - element not found.

Example:

```
//--- example for CArrayShort::SearchGreat(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100)!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   short  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchLess(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchGreatOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100)!=-1) printf("Element found");
   else                                  printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchLessOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100)!=-1) printf("Element found");
   else                                 printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   short  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchFirst(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100)!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayShort::SearchLast(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   short  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayShort::SearchLinear(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayShort::Save(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayShort::Load(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %d",i,array.At(i));
      }
    delete array;
   }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayShort - 82).

Example:

```
//--- example for CArrayShort::Type()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayInt

CArrayInt is a class of dynamic array of int or uint variables.

Description

The class CArrayInt provides the ability to work with a dynamic array of int or uint variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayInt : public CArray
```

Title

```
#include <Arrays\ArrayInt.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayInt

Direct descendants

- CSpreadBuffer

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

Memory control	
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size       // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayInt::Reserve(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size       // number
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayInt::Resize(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayInt::Shutdown()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   int   element       // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayInt::Add(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const int&   src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayInt::AddArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayInt*   src      // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayInt class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayInt::AddArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
  //--- use array
  //--- . . .
  //--- delete array
  delete array;
 }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   int  element,      // element to insert
   int  pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayInt::Insert(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const int&   src[],      // source array
   int           pos         // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayInt::InsertArray(const int &[],int)
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayInt*  src,      // pointer to the source
   int          pos       // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements to insert.

pos

[in] Position in the array to insert.

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayInt::InsertArray(const CArrayInt*,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
//--- delete source array
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const int&   src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayInt::AssignArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayInt*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayInt::AssignArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src  =new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int   pos,          // position
   int   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change.

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayInt::Update(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,10000))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // position
   int  shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayInt::Shift(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos      // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayInt::Delete(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of the last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayInt::DeleteRange(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
int  At(
   int  pos       // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, INT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, INT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayInt::At(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      int result=array.At(i);
      if(result==INT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const int&   src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayInt::CompareArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const CArrayInt*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayInt class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayInt::CompareArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   int  element       // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayInt::InsertSort(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(10000))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   int  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::Search(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(10000)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   int  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchGreat(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(10000)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   int  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLess(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(10000)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   int  element        // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchGreatOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(10000)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   int  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLessOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(10000)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   int  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt:: SearchFirst(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(10000)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   int  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayInt::SearchLast(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(10000)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   int  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayInt::SearchLinear(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(10000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayInt::Save(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayInt::Load(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int        file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %d",i,array.At(i));
     }
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayInt - 82).

Example:

```
//--- example for CArrayInt::Type()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayLong

CArrayLong class is a class of dynamic array of long or ulong variables.

Description

The CArrayLong class provides the ability to work with a dynamic array of long or ulong variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayLong : public CArray
```

Title

```
#include <Arrays\ArrayLong.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayLong

Direct descendants

CRealVolumeBuffer, CTickVolumeBuffer, CTimeBuffer

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

Memory control	
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayLong::Reserve(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size       // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayLong::Resize(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayLong::Shutdown()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   long  element      // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayLong::Add(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const long&  src[]      // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayLong::AddArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayLong*  src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayLong class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayLong::AddArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   long  element,      // element to insert
   int   pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into the array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayLong::Insert(long,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const long&  src[],      // source array
   int           pos         // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayLong::InsertArray(const long &[],int)
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayLong*  src,      // pointer to the source
   int           pos       // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayLong::InsertArray(const CArrayLong*,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
     return;
    }
//--- delete source array
 delete src;
//--- use array
//--- . . .
//--- delete array
 delete array;
}
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const long&  src[]      // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayLong::AssignArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayLong*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayLong::AssignArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src  =new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int    pos,          // position
   long   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayLong::Update(int,long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,1000000))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // position
   int  shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayLong::Shift(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos       // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayLong::Delete(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of the last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayLong::DeleteRange(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
long  At(
   int  pos       // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, LONG_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, LONG_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayLong::At(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      long result=array.At(i);
      if(result==LONG_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- Error reading from the array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const long&   src[]      // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayLong::CompareArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArrayconst

Compares the array with another one.

```
bool  CompareArrayconst(
   const CArrayLong*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayLong class used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayLong::CompareArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   long  element       // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayLong::InsertSort(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(1000000))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   long   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::Search(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(1000000)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   long  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchGreat(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(1000000)!=-1) printf("Element found");
   else                               printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   long  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLess(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(1000000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   long  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchGreatOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(1000000)!=-1) printf("Element found");
   else                                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   long   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLessOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(1000000)!=-1) printf("Element found");
   else                                     printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   long  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchFirst(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(1000000)!=-1) printf("Element found");
   else                               printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   long  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayLong::SearchLast(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(1000000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   long  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayLong::SearchLinear(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(1000000)!=-1) printf("Element found");
   else                                printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayLong::Save(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
      }
    delete array;
   }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayLong::Load(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %I64",i,array.At(i));
     }
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayLong - 84).

Example:

```
//--- example for CArrayLong::Type()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayFloat

CArrayFloat class is a class of dynamic array of float variables.

Description

The CArrayFloat class provides the ability to work with a dynamic array of float variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayFloat : public CArray
```

Title

```
#include <Arrays\ArrayFloat.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayFloat

Class Methods by Groups

Attributes	
Delta	Sets the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another

Attributes	
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Delta

Sets the comparison tolerance.

```
void  Delta(
   float   delta       // tolerance
   )
```

Parameters

delta

[in] The new value of the comparison tolerance.

Return Value

None

Note

Comparison tolerance is used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
//--- example for CArrayFloat::Delta(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set compare variation
   array.Delta(0.001);
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int   size      // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayFloat::Reserve(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size       // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayFloat::Resize(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayFloat::Shutdown()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   float   element       // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayFloat::Add(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const float&   src[]      // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayFloat::AddArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayFloat*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayFloat class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayFloat::AddArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   float  element,      // element to insert
   int    pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayFloat::Insert(float,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const float&  src[],      // source array
   int            pos        // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayFloat::InsertArray(const float &[],int)
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayFloat*  src,      // pointer to the source
   int            pos      // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayFloat::InsertArray(const CArrayFloat*,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
       return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const float&  src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayFloat::AssignArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayFloat*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayFloat::AssignArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src  =new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int     pos,          // position
   float   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayFloat::Update(int,float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100.0))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // position
   int  shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayFloat::Shift(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos      // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayFloat::Delete(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayFloat::DeleteRange(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
float  At(
   int  pos      // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, FLT_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, FLT_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayFloat::At(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      float result=array.At(i);
      if(result==FLT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const float&  src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayFloat::CompareArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

AssignArrayconst

Compares the array with another one.

```
bool  AssignArrayconst(
   const CArrayFloat*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayFloat class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayFloat::CompareArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   float   element      // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayFloat::InsertSort(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100.0))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   float  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::Search(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100.0)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchGreat(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat:: SearchLess(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchGreatOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchLessOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   float  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchFirst(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayFloat::SearchLast(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   float  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayFloat::SearchLinear(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100.0)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayFloat::Save(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayFloat::Load(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %f",i,array.At(i));
     }
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayFloat - 87).

Example:

```
//--- example for CArrayFloat::Type()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayDouble

CArrayDouble class is a class of dynamic array of double variables.

Description

The CArrayDouble class provides the ability to work with a dynamic array of double variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayDouble : public CArray
```

Title

```
#include <Arrays\ArrayDouble.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble

Direct descendants

- CDoubleBuffer

Class Methods by Groups

Attributes	
Delta	Set the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position

Attributes	
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified array position
Shift	Moves an element from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Search for min/max	
Minimum	Gets the lowest value index in the specified range
Maximum	Gets the highest value index in the specified range
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array

Attributes	
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Delta

Sets the comparison tolerance.

```
void  Delta(
   double  delta       // tolerance
   )
```

Parameters

delta

[in] The new value of the comparison tolerance.

Return Value

No

Note

Comparison tolerance is used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
//--- example for CArrayDouble::Delta(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set compare variation
   array.Delta(0.001);
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size       // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayDouble::Reserve(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size       // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayDouble::Resize(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayDouble::Shutdown()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   double  element      // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayDouble::Add(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const double&  src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayDouble::AddArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayDouble*  src      // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayDouble class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayDouble::AddArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   double  element,      // element to insert
   int     pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayDouble::Insert(double,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const double&  src[],      // source array
   int            pos         // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayDouble::InsertArray(const double &[],int)
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayDouble*  src,      // pointer to the source
   int            pos       // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayDouble::InsertArray(const CArrayDouble*,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
//--- delete source array
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const double&  src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayDouble::AssignArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayDouble*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayDouble::AssignArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src  =new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
    delete src;
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
   }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int      pos,          // position
   double   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element.

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayDouble::Update(int,double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100.0))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // position
   int  shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayDouble::Shift(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos       // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayDouble::Delete(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of the last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayDouble::DeleteRange(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
double  At(
   int  pos       // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, DBL_MAX - there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, DBL_MAX may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayDouble::At(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      double result=array.At(i);
      if(result==DBL_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //---  Error reading from the array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const double&   src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayDouble::CompareArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const CArrayDouble*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayDouble::CompareArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

Minimum

Gets index of the lowest element of the array in the specified range.

```
int  Minimum(
   int   start,      // starting index
   int   count       // number of elements
   ) const
```

Parameters

start

[in] Starting index of the search range.

count

[in] Search range size (number of elements).

Return Value

Index of the lowest element in the specified range.

Maximum

Gets index of the highest element of the array in the specified range.

```
int  Maximum(
   int   start,      // starting index
   int   count       // number of elements
   ) const
```

Parameters

start

[in] Starting index of the search range.

count

[in] Search range size (number of elements).

Return Value

Index of the highest element in the specified range.

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   double  element      // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayDouble::InsertSort(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100.0))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   double   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::Search(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100.0)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   double  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchGreat(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   double  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble:: SearchLess(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   double  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchGreatOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   double  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchLessOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   double  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchFirst(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   double  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayDouble::SearchLast(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   double  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayDouble::SearchLinear(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100.0)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayDouble::Save(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
      }
//--- delete array
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayDouble::Load(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %f",i,array.At(i));
     }
   //--- delete array
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayDouble - 87).

Example:

```
//--- example for CArrayDouble::Type()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayString

CArrayString class is a class of dynamic array of string variables.

Description

The CArrayString class provides the ability to work with a dynamic array of string variables. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

Declaration

```
class CArrayString : public CArray
```

Title

```
#include <Arrays\ArrayString.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayString

Class Methods by Groups

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds elements of one array to the end of another
AddArray	Adds elements of one array to the end of another
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
AssignArray	Copies the elements of one array to another
Update methods	

Memory control	
Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
CompareArray	Compares the array with another one
Sorted array opetations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from the file
virtual Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size       // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayString::Reserve(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. To reduce fragmentation of memory, the array size is changed by the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayString::Resize(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true - successful, false - error.

Example:

```
//--- example for CArrayString::Shutdown()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   string  element      // element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Example:

```
//--- example for CArrayString::Add(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(IntegerToString(i)))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const string&  src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayString::AddArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements of one array to the end of another.

```
bool  AddArray(
   const CArrayString*  src      // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of CArrayString class used as a source of elements to add.

Return Value

true - successful, false - cannot add items.

Example:

```
//--- example for CArrayString::AddArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   string  element,      // element to insert
   int     pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayString::Insert(string,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(IntegerToString(i),0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const string&  src[],      // source array
   int             pos         // position
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to insert

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayString::InsertArray(const string &[],int)
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   CArrayString*   src,      // pointer to the source
   int              pos       // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Example:

```
//--- example for CArrayString::InsertArray(const CArrayString*,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
//--- delete source array
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const string&  src[]       // source array
   )
```

Parameters

src[]

[in] Reference to an array used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayString::AssignArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the elements of one array to another.

```
bool  AssignArray(
   const CArrayString*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Example:

```
//--- example for CArrayString::AssignArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src  =new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int      pos,          // position
   string   element       // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Example:

```
//--- example for CArrayString::Update(int, string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,"ABC"))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,         // position
   int  shift        // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayString::Shift(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos      // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Example:

```
//--- example for CArrayString::Delete(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int   from,      // position of the first element
   int   to         // position of last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove the elements.

Example:

```
//--- example for CArrayString::DeleteRange(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
string  At(
   int  pos      // position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - success, ""- there was an attempt to get an element from a non-existing position (the last error code is ERR_OUT_OF_RANGE).

Note

Of course, "" may be a valid value of an array element. Therefore, always check the last error code after receiving such a value.

Example:

```
//--- example for CArrayString::At(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      string result=array.At(i);
      if(result=="" && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- Error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const string&   src[]       // source array
   ) const
```

Parameters

src[]

[in] Reference to an array used as a source of elements for comparison.

Return Value

true - arrays are equal, false - arrays are not equal.

Example:

```
//--- example for CArrayString::CompareArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArrays(
   const CArrayString*   src        // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayString class used as a source of elements for comparison.

Return Value

true - successful, false - cannot copy the items.

Example:

```
//--- example for CArrayString::CompareArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   string  element       // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Example:

```
//--- example for CArrayString::InsertSort(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort("ABC"))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString::Search(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search("ABC")!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   string  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString::SearchGreat(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat("ABC")!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchLess(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess("ABC")!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchGreatOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual("ABC")!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchLessOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual("ABC")!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchFirst(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst("ABC")!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   string  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayString:: SearchLast(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast("ABC")!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   string  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayString::SearchLinear(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear("ABC")!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayString::Save(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int            file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(IntegerToString(i));
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayString::Load(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = '%s'",i,array.At(i));
     }
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayString - 89).

Example:

```
//--- example for CArrayString::Type()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayObj

CArrayObj class is a class of dynamic array of pointers to instances of CObject and its derived classes.

Description

Class CArrayObj provides the ability to work with a dynamic array of pointers to instances of CObject and its derived classes. This allows working both with multidimensional dynamic arrays of primitive data types and with data structures that have more complex organization of data.

The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

There are certain subtleties of the class CArrayObj.

Declaration

```
class CArrayObj : public CArray
```

Title

```
#include <Arrays\ArrayObj.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj

Direct descendants

CIndicators, CSeries

Class Methods by Groups

Attributes	
FreeMode	Gets the flag of memory management
FreeMode	Sets the flag of memory management
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with full deallocation of its memory (but not its elements).
Creating a new element	
virtual CreateElement	Creates a new array element in the specified position

Attributes	
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds an element to the end of the array
Insert	Inserts an element to the specified position in the array
InsertArray	Inserts to an array elements from another array from the specified position
AssignArray	Copies the elements of one array to another
Update methods	
Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Detach	Gets the element from the specified position and removes it from the array
Delete	Removes the element from the specified array position
DeleteRange	Deletes a group of elements from the specified array position
Clear	Removes all elements of the array without the release of the array memory
Access methods	
At	Gets the element from the specified array position
Compare methods	
CompareArray	Compares the array with another one
Sorted array operations	
InsertSort	Inserts an element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element with a value exceeding the value of the sample in the sorted array
SearchLess	Searches for an element with a value less than the value of the sample in the sorted array

Attributes	
SearchGreatOrEqual	Searches for an element with a value greater than or equal to the value of the sample in the sorted array
SearchLessOrEqual	Searches for an element with a value less than or equal to the value of the sample in the sorted array
SearchFirst	Searches for the first element equal to the sample in the sorted array
SearchLast	Searches for the last element equal to the sample in the sorted array
Input/output	
Save	Saves data array in the file
Load	Loads data array from the file
Type	Gets the type identifier of the array

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Arrays of the CObject class have practical application (including all classes of the Standard Library).

For example, consider the options for two-dimensional array:

```
#include <Arrays\ArrayDouble.mqh>
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int i,j;
   int first_size=10;
   int second_size=100;
//--- create array
   CArrayObj    *array=new CArrayObj;
   CArrayDouble *sub_array;
//---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
//--- create subarrays
   for(i=0;i<first_size;i++)
```

```
      {
       sub_array=new CArrayDouble;
       if(sub_array==NULL)
         {
          delete array;
          printf("Object create error");
          return;
         }
       //--- fill array
       for(j=0;j<second_size;j++)
         {
          sub_array.Add(i*j);
         }
       array.Add(sub_array);
      }
//--- create array OK
   for(i=0;i<first_size;i++)
      {
       sub_array=array.At(i);
       for(j=0;j<second_size;j++)
         {
          double element=sub_array.At(j);
          //--- use array element
         }
      }
   delete array;
  }
```

Subtleties

The class has a mechanism to control dynamic memory, so be careful when working with elements of the array.

Mechanism of memory management can be switched on/off using the method FreeMode (bool). By default, the mechanism is enabled.

Accordingly, there are two options for dealing with the CArrayObj class:

1. Mechanism of memory management is enabled. (default)

In this case, CArrayObj takes responsibility for releasing the memory used for the elements after their removal from the array. A custom program should not release the array elements.

Example:

```
   int i;
//--- create an array
   CArrayObj *array=new CArrayObj;
//--- fill array elements
   for(i=0;i<10;i++) array.Add(new CObject);
```

```
//--- do something
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- actions performed with the element
      . . .
     }
//--- remove the array with the elements
   delete array;
```

2. Mechanism of memory management is disabled.

In this case, CArrayObj is not responsible for deallocating of the elements' memory after their removal from the array. Besides, the user program must deallocate the array elements.

Example:

```
   int i;
//--- create an array
   CArrayObj *array=new CArrayObj;
//--- disable the mechanism of memory management
   array.FreeMode(false);
//--- fill array with elements
   for(i=0;i<10;i++) array.Add(new CObject);
//--- do something
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- actions performed with the element
      . . .
     }
//--- remove array elements
   while(array.Total()) delete array.Detach();
//--- remove empty array
   delete array;
```

FreeMode

Gets the flag of memory management.

```
bool  FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CArrayObj::FreeMode()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get free mode flag
   bool array_free_mode=array.FreeMode();
   //--- delete array
   delete array;
  }
```

FreeMode

Sets the flag of memory management.

```
void  FreeMode(
   bool   mode        // new flag
   )
```

Parameters

mode

[in] New value of the memory management flag.

Return Value

None.

Note

Setting the memory management flag is an important part in the CArrayObj class use. Since the array elements are pointers to dynamic objects, it is important to determine what to do with them when removing from the array.

If the flag is set, removing an element from the array, the element is automatically deleted by the delete operator. If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be deallocated by the program afterwards.

If the user program resets the flag of memory management, the user must understand his responsibility for the removal of the array before the termination of the program. Otherwise the memory allocated for elements by the new operator is not released.

For large amounts of data this could lead even to crash of your terminal.

If the user does not reset the memory management flag, there is another pitfall. When pointer elements in array are stored somewhere in the local variables, then removing the array will lead to a critical error and crash of the user program. By default, the memory management flag is set, i.e. the class of the array is responsible for freeing the memory elements.

Example:

```
//--- example for CArrayObj::FreeMode(bool)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reset free mode flag
   array.FreeMode(false);
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int   size      // number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true - successful, false - there was an attempt to request for an amount less than or equal to zero, or failed to increase the array.

Note

To reduce fragmentation of memory, the array size is changed using the step previously determined by the Step(int) method or the default step of 16.

Example:

```
//--- example for CArrayObj::Reserve(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size       // size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true - successful, false - there was an attempt to set the size less than zero.

Note

Changing the size of the array allows using the memory optimally. Excessive elements on the right are lost. The memory of the lost elements is released or not depending on the memory management mode.

To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayObj::Resize(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Clear

Removes all elements of the array without the release of the memory array.

```
void  Clear()
```

Return Value

No.

Note

If the memory management flag is enabled, the memory used for the deleted objects is released.

Example:

```
//--- example for CArrayObj::Clear()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- clear array
   array.Clear();
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with full deallocation of memory for it (but not for its elements).

```
bool  Shutdown()
```

Return Value

true - successful, false - an error occurred.

Note

If memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CArrayObj::Shutdown()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

CreateElement

Creates a new array element at the specified position.

```
bool  CreateElement(
   int   index      // position
   )
```

Parameters

index

[in] Position in which you want to create a new element.

Return Value

true - successful, false - cannot create an element.

Note

Method CreateElement (int) in class CArrayObj always returns false and does not perform any action. If necessary, the CreateElement(int) method should be implemented in a derived class.

Example:

```
//--- example for CArrayObj::CreateElement(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int          size=100;
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- fill array
   array.Reserve(size);
   for(int i=0;i<size;i++)
     {
      if(!array.CreateElement(i))
        {
         printf("Element create error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   CObject*   element       // element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true - successful, false - cannot add an element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
//--- example for CArrayObj::Add(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(new CObject))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds elements from one array to the end of another.

```
bool  AddArray(
   const CArrayObj *  src       // pointer to the source array
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayDouble class - source of elements to add.

Return Value

true - successful, false - cannot add items.

Note

Adding elements from array to array is actually copying the pointers. Therefore, when calling the method, there is a pitfall - there may be a pointer to a dynamic object in more than one variable.

```
//--- example
extern bool        make_error;
extern int         error;
extern CArrayObj *src;
//--- create a new instance CArrayObj
//--- default memory management is turned on
CArrayObj *array=new CArrayObj;
//--- add (copy) the elements from the source array
if(array!=NULL)
   bool result=array.AddArray(src);
if(make_error)
  {
   //--- perform erroneous actions
   switch(error)
     {
      case 0:
         //--- remove the source array without checking its memory management flag
         delete src;
         //--- result:
         //--- it is possible to address an element by invalid pointer in the receive
         break;
      case 1:
         //--- disable the mechanism of memory management in the source array
         if(src.FreeMode()) src.FreeMode(false);
         //--- but do not remove the source array
         //--- result:
         //--- after removing the receiver array, it is possible to address an element
         break;
      case 2:
         //--- disable the mechanism of memory management in the source array
         src.FreeMode(false);
         //--- disable the mechanism of memory management in the receiver array
         array.FreeMode(false);
         //--- result:
         //--- after the program termination, get a "memory leak"
         break;
        }
     }
   else
     {
      //--- disable the mechanism of memory management in the source array
      if(src.FreeMode()) src.FreeMode(false);
      //--- delete the source array
      delete src;
      //--- result:
      //--- addressing the receiver array element will be correct
      //--- deleting the receiver array will lead to deleting its elements
     }
```

Example:

```
//--- example for CArrayObj::AddArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
```

```
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
//--- delete source array without elements
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

Insert

Inserts an element to the specified position in the array.

```
bool  Insert(
   CObject*  element,      // element to insert
   int       pos           // position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert the element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
//--- example for CArrayObj::Insert(CObject*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(new CObject,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts elements of one array from the specified position of another array.

```
bool  InsertArray(
   const CArrayObj*   src,      // pointer to the source
   int                pos       // position
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true - successful, false - cannot insert items.

Note

See: CArrayObj::AddArray(const CArrayObj*).

Example:

```
//--- example for CArrayObj::InsertArray(const CArrayObj*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array without elements
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Compares the array with another one.

```
bool  AssignArray(
   const CArrayObj*   src       // pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements to copy.

Return Value

true - successful, false - cannot copy the elements.

Note

If the receiver array is not empty when calling AssignArray, then all its elements will be removed; and if the memory management flag is set, the memory used for the deleted elements will be released. The receiver array becomes an exact copy of the source one. Additionally, see CArrayObj::AddArray(const CArrayObj*).

Example:

```
//--- example for CArrayObj::AssignArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
```

```
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
//--- arrays is identical
//--- delete source array without elements
   delete src;
//--- use array
//--- . . .
//--- delete array
   delete array;
  }
```

Update

Changes the element at the specified array position.

```
bool  Update(
   int        pos,         // position
   CObject*   element      // value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value of the element

Return Value

true - successful, false - cannot change the element.

Note

The element will not change if an invalid pointer (for example, NULL) is passed as a parameter. If the memory management is enabled, the memory of a replaced element is deallocated.

Example:

```
//--- example for CArrayObj::Update(int,CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,new CObject))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int   pos,        // position
   int   shift       // shift
   )
```

Parameters

pos

[in] Position of the moved element in the array

shift

[in] The shift value (both positive and negative).

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CArrayObj::Shift(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Detach

Removes an element from a given position in the array.

```
CObject*  Detach(
   int  pos       // position
   )
```

Parameters

pos

[in] Position of a removed item in the array.

Return Value

Pointer to the removed element - success, NULL - cannot remove the element.

Note

When an element is removed from the array, it will not be deleted regardless of the memory management flag. Once the array element pointer is used, it has to be deallocated.

Example:

```
//--- example for CArrayObj::Detach(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   CObject *object=array.Detach(0);
   if(object==NULL)
     {
      printf("Detach error");
      delete array;
      return;
     }
   //--- use element
   //--- . . .
   //--- delete element
   delete object;
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the specified array position.

```
bool  Delete(
   int  pos      // position
   )
```

Parameters

pos

[in] Position of the array element to be removed.

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CArrayObj::Delete(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from the specified array position.

```
bool  DeleteRange(
   int  from,      // position of the first element
   int  to         // position of last element
   )
```

Parameters

from

[in] Position of the first array element to be removed.

to

[in] Position of the last array element to be removed.

Return Value

true - successful, false - cannot remove elements.

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CArrayObj::DeleteRange(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the specified array position.

```
CObject*  At(
   int  pos      // position
   )
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element - successful, NULL- there was an attempt to get an element of a non-existent position.

Example:

```
//--- example for CArrayObj::At(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      CObject *result=array.At(i);
      if(result==NULL)
        {
         //--- Error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   delete array;
  }
```

CompareArray

Compares the array with another one.

```
bool  CompareArray(
   const CArrayObj*   src       // pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of the CArrayObj class used as a source of elements for comparison.

Return Value

true - the arrays are equal - the arrays are not equal.

Example:

```
//--- example for CArrayObj::CompareArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- fill source array
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts an element in a sorted array.

```
bool  InsertSort(
   CObject*   element       // element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true - successful, false - cannot insert the element.

Note

Element is not added to the array if an invalid pointer (such as NULL) is passed as a parameter.

Example:

```
//--- example for CArrayObj::InsertSort(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(new CObject))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   CObject*   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::Search(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.Search(sample)!=-1) printf("Element found");
   else                         printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element with a value exceeding the value of the sample in the sorted array.

```
int  SearchGreat(
   CObject*  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::SearchGreat(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchGreat(sample)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element with a value less than the value of the sample in the sorted array.

```
int  SearchLess(
   CObject*  element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj:: SearchLess(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLess(sample)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element with a value greater than or equal to the value of the sample in the sorted array.

```
int  SearchGreatOrEqual(
   CObject*   element       // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::SearchGreatOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchGreatOrEqual(sample)!=-1) printf("Element found");
   else                                     printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element with a value less than or equal to the value of the sample in the sorted array.

```
int  SearchLessOrEqual(
   CObject*   element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj:: SearchLessOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLessOrEqual(sample)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Searches for the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   CObject*  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj::SearchFirst(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchFirst(sample)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Searches for the last element equal to the sample in the sorted array.

```
int  SearchLast(
   CObject*  element      // sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element - successful, -1 - the element not found.

Example:

```
//--- example for CArrayObj:: SearchLast(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLast(sample)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CArrayObj::Save(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from the file.

```
virtual bool  Load(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen(...) function.

Return Value

true - successfully completed, false - error.

Note

When reading array elements from the file, the CArrayObj::CreateElement(int) method is called to create each element.

Example:

```
//--- example for CArrayObj::Load(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   //--- . . .
   //--- delete array
   delete array;
  }
```

Type

Gets the array type identifier.

```
virtual int  Type() const
```

Return Value

Array type identifier (for CArrayObj - 7778).

Example:

```
//--- example for CArrayObj::Type()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CList

CList Class is a class of dynamic list of instances of the CObject class and its derived classes.

Description

Class CList provides the ability to work with a list of instances of CObject and its derived classes. The class allows adding/inserting/deleting array elements, performing an array sorting, and searching in a sorted array. In addition, methods of working with files have been implemented.

There are some subtleties of working with the CList class. The class has a mechanism to control dynamic memory, so be careful when working with elements of the list.

Subtleties of the mechanism of memory management similar to those described in CArrayObj.

Declaration

```
   class CList : public CObject
```

Title

```
   #include <Arrays\List.mqh>
```

Inheritance hierarchy

CObject
    CList

Class Methods by Groups

Attributes	
FreeMode	Gets the flag of memory management when deleting list elements
FreeMode	Sets the flag of memory management when deleting list elements
Total	Gets the number of elements in the list
IsSorted	Gets sorted list flag
SortMode	Gets the sorting mode
Create methods	
CreateElement	Creates a new list element
Add methods	
Add	Adds an element to the end of the list
Insert	Inserts an element to the specified position of the list
Delete methods	
DetachCurrent	Removes an element from the current position in the list without deleting it "physically"

Attributes	
DeleteCurrent	Removes the element from the current position in the list
Delete	Removes the element from the specified position in the list
Clear	Removes all list elements
Navigation	
IndexOf	Gets the index of the specified list element
GetNodeAtIndex	Gets an item with the specified index of the list
GetFirstNode	Gets the first element of the list
GetPrevNode	Gets the previous element of the list
GetCurrentNode	Gets the current list element
GetNextNode	Gets the next element in the list
GetLastNode	Gets the last element in the list
Ordering methods	
Sort	Sorts the list
MoveToIndex	Moves the current element in the list to the specified position
Exchange	Swaps two elements in the list
Compare methods	
CompareList	Compares the list with another one
Search methods	
Search	Searches for an element equal to the sample in sorted list
Input/output	
virtual Save	Saves list data in the file
virtual Load	Loads list data from the file
virtual Type	Gets the list type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode

Gets the flag of memory management when deleting list elements.

```
bool  FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CList::FreeMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get free mode flag
   bool list_free_mode=list.FreeMode();
   //--- delete list
   delete list;
  }
```

FreeMode

Sets the flag of memory management when deleting list elements.

```
void  FreeMode(
   bool  mode        // new value
   )
```

Parameters

mode

[in] New value of the memory management flag.

Note

Setting the memory management flag is an important part in the CList class use. Since the list elements are pointers to dynamic objects, it is important to determine what to do with them when removing from the list.

If the flag is set, when removing an element from the list, the element is automatically deleted by the delete operator. If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be deallocated by the program afterwards.

If the user program resets the flag of memory management, users should understand their responsibility for the removal of the list elements before the termination of the program. Otherwise, the memory allocated for elements by the new operator is not released.

For large amounts of data this could lead even to a terminal crash.

If the user program does not reset the memory management flag, there is another "pitfall". When pointer elements in list are stored somewhere in the local variables, then removing the list will lead to a critical error and crash of the user program. By default, the memory management flag is set, i.e. the class of the list is responsible for releasing the memory elements.

Example:

```
//--- example for CList::FreeMode(bool)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reset free mode flag
   list.FreeMode(false);
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Total

Gets the number of elements in the list.

```
int  Total() const
```

Return Value

Number of elements in the list.

Example:

```
//--- example for CList::Total()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check total
   int total=list.Total();
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

IsSorted

Gets the sorted list flag.

```
bool  IsSorted(
   int   mode=0       // sorting mode
   ) const
```

Parameters

mode=0

[in] Checked sort mode.

Return Value

Flag of the sorted list. Returns true if the list is sorted using the specified mode, otherwise returns false.

Note

Flag of the sorted list cannot be changed directly. The flag is set by Sort(int) and resets by any add/insert methods.

Example:

```
//--- example for CList::IsSorted()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sorted
   if(list.IsSorted(0))
     {
      //--- use methods for sorted list
      //--- ...
     }
   //--- delete list
   delete list;
  }
```

SortMode

Gets the version of the sorting.

```
int  SortMode() const
```

Return Value

Sorting mode, or -1 if the list is not sorted.

Example:

```
//--- example for CList::SortMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sort mode
   int sort_mode=list.SortMode();
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

CreateElement

Creates a new element of the list.

```
CObject*  CreateElement()
```

Return Value

Pointer to the newly created element - successful, NULL - cannot create an element.

Note

Method CreateElement () in the CList class always returns NULL and does not perform any actions. If necessary, method CreateElement () should be implemented in a derived class.

Example:

```
//--- example for CList::CreateElement(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int     size=100;
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- fill list
   for(int i=0;i<size;i++)
     {
      CObject *object=list.CreateElement();
      if(object==NULL)
        {
         printf("Element create error");
         delete list;
         return;
        }
      list.Add(object);
     }
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Add

Adds an element to the end of the list.

```
int  Add(
   CObject*   element       // element to add
   )
```

Parameters

element

[in] Value of the element to add to the list.

Return Value

Index of the added element - success, -1 - error.

Note

The element is not added to the list, if an invalid pointer (for example, NULL) is passed as a parameter.

Example:

```
//--- example for CList::Add(Cobject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 elements
   for(int i=0;i<100;i++)
     {
      if(list.Add(new CObject)==-1)
        {
         printf("Element addition error");
         delete list;
         return;
        }
     }
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Insert

Inserts an element to the specified position in the list.

```
int  Insert(
   CObject*   element,      // element to insert
   int        pos           // position
   )
```

Parameters

element

[in] value of the element to insert in the list

pos

[in] position in the list to insert

Return Value

index of inserted element - success, -1 - error.

Note

The element is not added to the list, if an invalid pointer (for example, NULL) is passed as a parameter.

Example:

```
//--- example for CList::Insert(CObject*,int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert 100 elements
   for(int i=0;i<100;i++)
     {
      if(list.Insert(new CObject,0)==-1)
        {
         printf("Element insert error");
         delete list;
         return;
        }
     }
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

DetachCurrent

Extracts an element from the current position in the list without its "physical" deletion.

```
CObject*  DetachCurrent()
```

Return Value

Pointer to the removed element in case of success, NULL - cannot remove the element.

Note

When removed from the list, the element is not removed in any state of the memory management flag. The pointer to the extracted element should be released after it has been used.

Example:

```
//--- example for CList::DetachCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.DetachCurrent();
   if(object==NULL)
     {
      printf("Detach error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- delete element
   delete object;
   //--- delete list
   delete list;
  }
```

DeleteCurrent

Removes the element from the current position in the list.

```
bool  DeleteCurrent()
```

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management is enabled, memory for the removed element is deallocated.

Example:

```
//--- example for CList::DeleteCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   if(!list.DeleteCurrent())
     {
      printf("Delete error");
      delete list;
      return;
     }
   //--- delete list
   delete list;
  }
```

Delete

Removes the element from the given position in the list.

```
bool  Delete(
   int  pos       // position
   )
```

Parameters

pos

[in] position of element to be removed from the list.

Return Value

true - successful, false - cannot remove the element.

Note

If the memory management flag is enabled, the memory used for the deleted element is released.

Example:

```
//--- example for CList::Delete(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   if(!list.Delete(0))
     {
      printf("Delete error");
      delete list;
      return;
     }
   //--- delete list
   delete list;
  }
```

Clear

Removes all elements of the list.

```
void  Clear()
```

Note

If the memory management is enabled, the memory of deleted elements is deallocated.

Example:

```
//--- example for CList::Clear()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   //--- clear list
   list.Clear();
   //--- delete list
   delete list;
  }
```

IndexOf

Gets the index of the specified list element.

```
int  IndexOf(
   CObject*   element       // pointer to the element
   )
```

Parameters

element

[in] pointer to the list element.

Return Value

List element index, or -1.

Example:

```
//--- example for CList::IndexOf(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   CObject *object=new CObject;
   if(object==NULL)
     {
      printf("Element create error");
      delete list;
      return;
     }
   if(list.Add(object))
     {
      int pos=list.IndexOf(object);
     }
   //--- delete list
   delete list;
  }
```

GetNodeAtIndex

Gets an element from the specified position in the list.

```
CObject*  GetNodeAtIndex(
   int  pos       // position
   )
```

Parameters

pos

[in] element position in the list.

Return Value

pointer to the element - success, NULL - cannot receive a pointer.

Example:

```
//--- example for CList::GetNodeAtIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetNodeAtIndex(10);
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetFirstNode

Gets the first element of the list.

```
CObject*  GetFirstNode()
```

Return Value

Pointer to the first element - success, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetFirstNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetFirstNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetPrevNode

Gets the previous element of the list.

```
CObject*  GetPrevNode()
```

Return Value

Pointer to the previous element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetPrevNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetPrevNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetCurrentNode

Gets the current list element.

```
CObject*  GetCurrentNode()
```

Return Value

Pointer to the current element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetCurrentNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetCurrentNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetNextNode

Gets the next element in the list.

```
CObject*  GetNextNode()
```

Return Value

Pointer to the next element - successful, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetNextNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetNextNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetLastNode

Gets the last element of the list.

```
CObject*  GetLastNode()
```

Return Value

Pointer to the last element - success, NULL - cannot get a pointer.

Example:

```
//--- example for CList::GetLastNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetLastNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

Sort

Sorts a list.

```
void  Sort(
   int   mode         // sorting mode
   )
```

Parameters

mode

[in] Sorting mode.

Return Value

No.

Note

The list is always sorted in ascending order.

Example:

```
//--- example for CList::Sort(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- sorting by mode 0
   list.Sort(0);
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

MoveToIndex

Moves the current element in the list to the specified position.

```
bool  MoveToIndex(
   int  pos      // position
   )
```

Parameters

pos

[in] position in the list to move.

Return Value

true - successful, false - cannot move the element.

Example:

```
//--- example for CList::MoveToIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- move current node to begin
   list.MoveToIndex(0);
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Exchange

Swaps two elements in the list.

```
bool  Exchange(
   CObject*   node1,      // list element
   CObject*   node2       // list element
   )
```

Parameters

node1

[in] list element

node2

[in] list element

Return Value

true - successful, false - cannot swap the elements.

Example:

```
//--- example for CList::Exchange(CObject*,CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- exchange
   list.Exchange(list.GetFirstNode(),list.GetLastNode());
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

CompareList

Compares the list with another one.

```
bool  CompareList(
   CList*  list        // pointer to the source
   )
```

Parameters

list

[in] a pointer to an instance of the CList class used as a source of elements for comparison.

Return Value

true - the lists are equal, false - the lists are not equal.

Example:

```
//--- example for CList::CompareList(const CList*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source list
   CList *src=new CList;
   if(src==NULL)
     {
      printf("Object create error");
      delete list;
      return;
     }
   //--- fill source list
   //--- . . .
   //--- compare with another list
   bool result=list.CompareList(src);
   //--- delete lists
   delete src;
   delete list;
  }
```

Search

Searches for an element equal to the sample in the sorted list.

```
CObject*  Search(
   CObject*  element      // sample
   )
```

Parameters

element

[in] element sample to search for in the list.

Return Value

Pointer to the found element - successful, NULL - the element is not found.

Example:

```
//--- example for CList::Search(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add lists elements
   //--- . . .
   //--- sort list
   list.Sort(0);
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete list;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(list.Search(sample)!=NULL) printf("Element found");
   else                          printf("Element not found");
   //--- delete list
   delete list;
  }
```

Save

Saves list data in the file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CList::Save(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add lists elements
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete list
   delete list;
  }
```

Load

Loads list data from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened using the FileOpen () function.

Return Value

true - successfully completed, false - error.

Note

When reading list elements from the file, the CList::CreateElement(int) method is called to create each element.

Example:

```
//--- example for CLoad::Load(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use list elements
   //--- . . .
   //--- delete list
   delete list;
  }
```

Type

Gets the list type identifier.

```
virtual int  Type()
```

Return Value

List type identifier (for CList - 7779).

Example:

```
//--- example for CList::Type()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get list type
   int type=list.Type();
   //--- delete list
   delete list;
  }
```

CTreeNode

CTreeNode is a class of the CTree binary tree node.

Description

CTreeNode provides the ability to work with nodes of the CTree binary tree. Options of navigation through the tree are implemented in the class. Besides, methods of working with the file are implemented.

Declaration

```
class CTreeNode : public CObject
```

Title

```
#include <Arrays\TreeNode.mqh>
```

Inheritance hierarchy

CObject
    CTreeNode

Direct descendants

CTree

Class Methods by Groups

Attributes	
Owner	Gets/sets the pointer of the owner node
Left	Gets/sets the pointer of the left node
Right	Gets/sets the pointer of the right node
Balance	Gets the node balance
BalanceL	Gets the balance of the left sub-branch of the node
BalanceR	Gets the balance of the right sub-branch of the node
Creation of a new element	
CreateSample	Creates a new node instance
Comparison	
RefreshBalance	Recalculates the node balance
Search	
GetNext	Gets the pointer of the next node
Input/Output	

Attributes	
SaveNode	Saves the node data to a file
LoadNode	Downloads the node data from a file
virtual Type	Gets the identifier of the node type

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Compare

Trees of CTreeNode class descendants have practical application.

A descendant of CTreeNode class should have predefined methods: CreateSample creates a new instance of the descendant class of CTreeNode, Compare compares values of key fields of the descendant class of CTreeNode, Type (if it is necessary to identify a node), SaveNode and LoadNode (if it is necessary to work with a file).

Let's consider an example of a CTree descendant class.

```
//+------------------------------------------------------------------+
//|                                                 MyTreeNode.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                             https://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\TreeNode.mqh>
//+------------------------------------------------------------------+
//| Describe CMyTreeNode class derived from CTreeNode.               |
//+------------------------------------------------------------------+
//| Class CMyTreeNode.                                               |
//| Purpose: Class of element of a binary tree.                      |
//|             Descendant of class CTreeNode.                       |
//+------------------------------------------------------------------+
class CMyTreeNode : public CTreeNode
  {
protected:
   //--- user's data
   long              m_long;           // key field of long type
   double            m_double;         // custom variable of double type
   string            m_string;         // custom variable of string type
   datetime          m_datetime;       // custom variable of datetime type

public:
                     CMyTreeNode();
   //--- methods of accessing user's data
   long              GetLong(void)               { return(m_long); }
   void              SetLong(long value)         { m_long=value;  }
   double             GetDouble(void)             { return(m_double); }
   void               SetDouble(double value)     { m_double=value;   }
   string             GetString(void)             { return(m_string); }
   void               SetString(string value)     { m_string=value;   }
   datetime           GetDateTime(void)           { return(m_datetime); }
   void               SetDateTime(datetime value) { m_datetime=value;   }
   //--- methods for working with files
   virtual bool       Save(int file_handle);
   virtual bool       Load(int file_handle);
protected:
   virtual int        Compare(const CObject *node,int mode);
   //--- method of creating class instances
   virtual CTreeNode* CreateSample();
  };
//+------------------------------------------------------------------+
//| CMyTreeNode class constructor.                                   |
//| INPUT:  none.                                                    |
//| OUTPUT: none.                                                    |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
void CMyTreeNode::CMyTreeNode()
  {
//--- initialization of user's data
   m_long        =0;
   m_double      =0.0;
   m_string      ="";
   m_datetime    =0;
  }
//+------------------------------------------------------------------+
//| Comparison with another tree node by the specified algorithm.    |
//| INPUT:  node - tree element to compare,                          |
//|         mode - identifier of comparison algorithm.               |
//| OUTPUT: result of comparison (>0,0,<0).                          |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
int CMyTreeNode::Compare(const CObject *node,int mode)
  {
//--- mode parameter is ignored, because tree construction algorithm is the only one
   int res=0;
//--- explicit type casting
   CMyTreeNode *n=node;
   res=(int)(m_long-n.m_long);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Creation of a new class instance.                                |
//| INPUT:  none.                                                    |
//| OUTPUT: pointer to a new instance of CMyTreeNode class.          |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CTreeNode* CMyTreeNode::CreateSample()
  {
   CMyTreeNode *result=new CMyTreeNode;
//---
   return(result);
  }
//+------------------------------------------------------------------+
//| Write tree node data to a file.                                  |
//| INPUT:  file_handle -handle of a file pre-opened for writing.    |
//| OUTPUT: true if OK, otherwise false.                             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
bool CMyTreeNode::Save(int file_handle)
  {
   uint i=0,len;
//--- checks
   if(file_handle<0) return(false);
//--- writing user data
//--- writing custom variable of long type
   if(FileWriteLong(file_handle,m_long)!=sizeof(long))         return(false);
//--- writing custom variable of double type
   if(FileWriteDouble(file_handle,m_double)!=sizeof(double))    return(false);
//--- writing custom variable of string type
   len=StringLen(m_string);
//--- write string length
   if(FileWriteInteger(file_handle,len,INT_VALUE)!=INT_VALUE)   return(false);
//--- write the string
   if(len!=0 && FileWriteString(file_handle,m_string,len)!=len) return(false);
//--- writing custom variable of datetime type
   if(FileWriteLong(file_handle,m_datetime)!=sizeof(long))      return(false);
//---
   return(true);
  }
//+------------------------------------------------------------------+
//| Read tree node data from a file.                                 |
//| INPUT:  file_handle -handle of a file pre-opened for reading.    |
//| OUTPUT: true if OK, otherwise false.                             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
bool CMyTreeNode::Load(int file_handle)
  {
   uint i=0,len;
//--- checks
   if(file_handle<0) return(false);
//--- reading
   if(FileIsEnding(file_handle)) return(false);
//--- reading custom variable of char type
//--- reading custom variable of long type
   m_long=FileReadLong(file_handle);
//--- reading custom variable of double type
   m_double=FileReadDouble(file_handle);
//--- reading custom variable of string type
//--- read the string length
   len=FileReadInteger(file_handle,INT_VALUE);
//--- read the string
   if(len!=0) m_string=FileReadString(file_handle,len);
   else       m_string="";
//--- reading custom variable of datetime type
   m_datetime=FileReadLong(file_handle);
//---
   return(true);
  }
```

Owner

Gets the pointer of the owner node.

```
CTreeNode*  Owner()
```

Return Value

Pointer of the owner node.

Owner

Sets the pointer of the owner node.

```
void  Owner(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] New value of the pointer of the owner node.

Return Value

None.

Left

Gets the pointer of the left node.

```
CTreeNode*  Left()
```

Return Value

Pointer of the left node.

Left

Sets the pointer of the left node.

```
void  Left(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] New value of the pointer of the left node.

Return Value

None.

Right

Gets the pointer of the right node.

```
CTreeNode*  Right()
```

Return Value

The pointer of the right node.

Right

Sets the pointer of the right node.

```
void  Right(
   CTreeNode*  node       // node
   )
```

Parameters

node

[in] New value of the pointer of the right node.

Return Value

None.

Balance

Gets the node balance.

```
int  Balance() const
```

Return Value

Node balance.

BalanceL

Gets the balance of the left sub-branch of the node.

```
int  BalanceL() const
```

Return Value

Balance of the left sub-branch of the node.

BalanceR

Gets the balance of the right sub-branch of the node.

```
int  BalanceR() const
```

Return Value

Balance of the right sub-branch of the node.

CreateSample

Creates a new node sample.

```
virtual CTreeNode*  CreateSample()
```

Return Value

Pointer to the new node sample or NULL.

RefreshBalance

Recalculates the node balance.

```
int  RefreshBalance()
```

Return Value

Node balance.

GetNext

Gets the pointer of the next node.

```
CTreeNode*  GetNext(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] Node of the search start.

Return Value

Pointer of the next node.

SaveNode

Writes node data to a file.

```
bool  SaveNode(
   int  file_handle       // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true - success, otherwise false.

LoadNode

Reads node data from a file.

```
bool  LoadNode(
   int          file_handle,      // handle
   CTreeNode*   main              // node
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

main

[in] Node for data.

Return Value

true - success, otherwise false.

Type

Gets the identifier of the node type.

```
virtual int  Type() const
```

Return Value

Identifier of the node type.

CTree

CTree is a class of the binary tree of the instances of CTreeNode class and its descendants.

Description

CTree class provides the possibility to work with the binary tree of CTreeNode class instances and its descendants. Options of adding/inserting/deleting of tree elements and search in the tree are implemented in the class. Besides that, methods of working with a file are implemented.

Note that mechanism of dynamic memory management is not implemented in class CTree (unlike classes CList and CArrayObj). All tree nodes are deleted with memory deallocation.

Declaration

```
class CTree : public CTreeNode
```

Title

```
#include <Arrays\Tree.mqh>
```

Inheritance hierarchy

- CObject
  - CTreeNode
    - CTree

Class Methods by Groups

Attributes	
Root	Gets the root node of the tree
Creation of a new element	
CreateElement	Creates a new instance of the node
Filling	
Insert	Adds a node to a tree
Deletion	
Detach	Detaches a specified node from a tree
Delete	Deletes a specified node from a tree
Clear	Deletes all nodes of a tree
Search	
Find	Searches for a node in a tree by sample
Input/output	

Attributes	
virtual Save	Saves all the tree data to a file
virtual Load	Downloads tree data from a file
virtual Type	Gets identifier of the tree type

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CTreeNode

Parent, Parent, Left, Left, Right, Right, Balance, BalanceL, BalanceR, RefreshBalance, GetNext, SaveNode, LoadNode

Trees of CTreeNode class descendants - descendants of class CTree - have practical application.

Descendant of CTree class should have a predefined method CreateElement that creates a new instance of the CTreeNode descendant class.

Let's consider an example of the CTree descendant class.

```
//+------------------------------------------------------------------+
//|                                                       MyTree.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                       https://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\Tree.mqh>
#include "MyTreeNode.mqh"
//---
input int extCountedNodes = 100;
//+------------------------------------------------------------------+
//| Describe class CMyTree derived from CTree.                       |
//+------------------------------------------------------------------+
//| Class CMyTree.                                                   |
//| Purpose: Construction and navigation of a binary search tree.    |
//+------------------------------------------------------------------+
class CMyTree : public CTree
  {
public:
   //--- methods of searching in the tree by custom data
   CMyTreeNode*        FindByLong(long find_long);
   //--- method of creation of the tree element
   virtual CTreeNode *CreateElement();
  };
//---
CMyTree MyTree;
```

```
//+------------------------------------------------------------------+
//| Creation of a new tree node.                                     |
//| INPUT:  none.                                                    |
//| OUTPUT: pointer to the new tree node if OK, or NULL.             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CTreeNode *CMyTree::CreateElement()
  {
   CMyTreeNode *node=new CMyTreeNode;
//---
   return(node);
  }
//+------------------------------------------------------------------+
//| Search of element in a list by value m_long.                     |
//| INPUT:  find_long - searched value.                              |
//| OUTPUT: pointer of a found list element, or NULL.                |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CMyTreeNode* CMyTree::FindByLong(long find_long)
  {
   CMyTreeNode *res=NULL;
   CMyTreeNode *node;
//--- create a tree node to pass the search parameter
   node=new CMyTreeNode;
   if(node==NULL) return(NULL);
   node.SetLong(find_long);
//---
   res=Find(node);
   delete node;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| script "testing of class CMyTree"                                |
//+------------------------------------------------------------------+
//---  array for string initialization
string str_array[11]={"p","oo","iii","uuuu","yyyyy","ttttt","rrrr","eee","ww","q","99
//---
int OnStart() export
  {
   int          i;
   uint         pos;
   int          beg_time,end_time;
   CMyTreeNode *node; //--- temporary pointer to the sample of class CMyTreeNode
//---
   printf("Start test %s.",__FILE__);
//--- Fill out MyTree with instances of class MyTreeNode in the amount of extCountedN
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i++)
```

```
     {
      node=MyTree.CreateElement();
      if(node==NULL)
        {
         //--- emergency exit
         printf("%s (%4d): create error",__FILE__,__LINE__);
         return(__LINE__);
        }
      NodeSetData(node,i);
      node.SetLong(i);
      MyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("Filling time of MyTree is %d ms.",end_time-beg_time);
//--- Create a temporary tree TmpMyTree.
   CMyTree TmpMyTree;
//--- Detach 50% of tree elements (all even)
//--- and add them to the temporary tree TmpMyTree.
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i+=2)
     {
      node=MyTree.FindByLong(i);
      if(node!=NULL)
         if(MyTree.Detach(node)) TmpMyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("Deletion time of %d elements from MyTree is %d ms.",extCountedNodes/2,end_t
//--- Return the detached
   node=TmpMyTree.Root();
   while(node!=NULL)
     {
      if(TmpMyTree.Detach(node)) MyTree.Insert(node);
      node=TmpMyTree.Root();
     }
//--- Check work of method Save(int file_handle);
   int file_handle;
   file_handle=FileOpen("MyTree.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!MyTree.Save(file_handle))
        {
         //--- error writing to a file
         //--- emergency exit
         printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
         //--- close file before leaving!!!
         FileClose(file_handle);
         return(__LINE__);
        }
      FileClose(file_handle);
     }
//--- Check work of method Load(int file_handle);
   file_handle=FileOpen("MyTree.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!TmpMyTree.Load(file_handle))
        {
         //--- error reading from file
         //--- emergency exit
         printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
         //--- close file before leaving!!!
         FileClose(file_handle);
         return(__LINE__);
        }
      FileClose(file_handle);
     }
//---
   MyTree.Clear();
   TmpMyTree.Clear();
//---
   printf("End test %s. OK!",__FILE__);
//---
   return(0);
  }
//+------------------------------------------------------------------+
//| Function to output node contents to journal                      |
//+------------------------------------------------------------------+
void NodeToLog(CMyTreeNode *node)
  {
   printf("   %I64d,%f,'%s','%s'",
               node.GetLong(),node.GetDouble(),
               node.GetString(),TimeToString(node.GetDateTime()));
  }
//+------------------------------------------------------------------+
//| Function to populate node with random values                     |
//+------------------------------------------------------------------+
void NodeSetData(CMyTreeNode *node,int mode)
  {
   if(mode%2==0)
     {
      node.SetLong(mode*MathRand());
      node.SetDouble(MathPow(2.02,mode)*MathRand());
     }
   else
     {
      node.SetLong(mode*(long)(-1)*MathRand());
      node.SetDouble(-MathPow(2.02,mode)*MathRand());
     }
   node.SetString(str_array[mode%10]);
```

```
    node.SetDateTime(10000*mode);
   }
```

Root

Gets the root node of the tree.

```
CTreeNode*  Root() const
```

Return Value

Pointer of the root node of the tree.

CreateElement

Creates a new instance of the node.

```
virtual CTreeNode*  CreateElement()
```

Return Value

Pointer of the new instance of the node or NULL.

Insert

Adds a node to a tree.

```
CTreeNode*  Insert(
   CTreeNode*   new_node      // node
   )
```

Parameters

new_node

[in] Pointer of a node to insert to a tree.

Return Value

Pointer of the owner node or NULL.

Detach

Detaches a specified node from a tree.

```
bool  Detach(
   CTreeNode*   node       // node
   )
```

Parameters

node

[in] Node pointer to detach.

Return Value

true - success, otherwise false.

Note

After detachment, the node pointer is not released. The tree is balanced.

Delete

Deletes a specified node from a tree.

```
bool  Delete(
   CTreeNode*   node       // node
   )
```

Parameters

node

[in] Node pointer to delete.

Return Value

true - success, otherwise false.

Note

After deletion, a node pointer is released. The tree is balanced.

Clear

Deletes all nodes of a tree.

```
void  Clear()
```

Return Value

None.

Note

After deletion, node pointers are released.

Find

Searches for a node in a tree by sample.

```
CTreeNode*  Find(
   CTreeNode*  node       // node
   )
```

Parameters

node

[in] Node that contains data used as a search sample.

Return Value

Pointer of the found node or NULL.

Save

Writes tree data to a file.

```
virtual bool  Save(
   int   file_handle       // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true - success, otherwise false.

Load

Reads tree data from a file.

```
virtual bool  Load(
   int   file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

Return Value

true - success, otherwise false.

Type

Gets identifier of the tree type.

```
virtual int  Type() const
```

Return Value

Identifier of the tree type.

Generic Data Collections

The library provides classes and interfaces that define generic collections, which allow users to create strongly typed collections. These collections provide greater convenience and data handling performance than non-generic typed collections.

The library is available in the Include\Generic folder of the terminal working directory.

Objects:

Object	Description	Type
ICollection	Interface for implementing generic data collections	INTERFACE
IEqualityComparable	Interface for implementing objects that can be compared	INTERFACE
IComparable	Interface for implementing objects that can be compared in terms of "greater than, less than or equal to"	INTERFACE
IComparer	Interface for implementing a generic class that compares two object of the T type, whether one is "greater than, less than or equal to" the other one	INTERFACE
IEqualityComparer	Interface for implementing a generic class that compares two object of the T type for equality	INTERFACE
IList	Interface for implementing generic data lists	INTERFACE
IMap	Interface for implementing generic collections of key/value pairs	INTERFACE
ISet	Interface for implementing generic data sets	INTERFACE
CDefaultComparer	A helper class that implements the IComparer<T> generic interface based on Compare global methods	CLASS
CDefaultEqualityComparer	A helper class that implements the IEqualityComparer<T> generic interface using Equals<T> and GetHashCode global methods	CLASS
CArrayList	A generic class that implements the IList<T> interface	CLASS
CKeyValuePair	The class implements the key/value pair	CLASS

Object	Description	Type
CHashMap	A generic class that implements the IMap<TKey, TValue> interface	CLASS
CHashSet	A generic class that implements the ISet<T> interface	CLASS
CLinkedListNode	A helper class for implementing the CLinkedListNode<T> class	CLASS
CLinkedList	A generic class that implements the ICollection<T> interface	CLASS
CQueue	A generic class that implements the ICollection<T> interface	CLASS
CRedBlackTreeNode	A helper class used in implementing the CRedBlackTree<T> class	CLASS
CRedBlackTree	A generic class that implements the ICollection<T> interface	CLASS
CSortedMap	A generic class that implements the IMap<TKey, TValue> interface	CLASS
CSortedSet	A generic class that implements the ISet<T> interface	CLASS
CStack	A generic class that implements the ICollection<T> interface	CLASS

Global methods:

Method	Description
ArrayBinarySearch	Searches for the specified value in an ascending-sorted one-dimensional array using the IComparable<T> interface to compare elements
ArrayIndexOf	Searches for the first occurrence of a value in a one-dimensional array
ArrayLastIndexOf	Searches for the last occurrence of a value in a one-dimensional array
ArrayReverse	Changes the sequence of elements in a one-dimensional array
Compare	Compares two values, whether one of them is greater than, less than or equal to the other one
Equals	Compares two values for equality
GetHashCode	Calculates the hash code value

ICollection<T>

ICollection<T> is an interface for implementing generic data collections.

Description

The ICollection<T> interface determines basic methods to work with collections, including methods to count elements, to clear a collection, to add or delete elements, and others.

Declaration

```
   template<typename T>
   interface ICollection
```

Header

```
   #include <Generic\Interfaces\ICollection.mqh>
```

Inheritance Hierarchy

ICollection

Direct descendants

CLinkedList, CQueue, CRedBlackTree, CStack, IList, IMap, ISet

Class Methods

Method	Description
Add	Adds an element to a collection
Count	Returns the number of elements in a collection
Contains	Determines whether a collection contains an element with the specified value
CopyTo	Copies all elements of a collection to the specified array starting at the specified index
Clear	Removes all elements from a collection
Remove	Removes the first occurrence of the specified element from a collection

Add

Adds an element to a collection.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in a collection.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether a collection contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the collection, or false otherwise.

CopyTo

Copies all elements of a collection to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the collection will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements of a collection.

```
void Clear();
```

Remove

Removes the first occurrence of the specified element from a collection.

```
bool Remove(
   T  item      // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

IEqualityComparable<T>

IEqualityComparable<T> is an interface for implementing objects that can be compared.

Description

The IEqualityComparable<T> interface defines methods to retrieve the hash code of the current object and to check whether it is equal to another object of the same type.

Declaration

```
   template<typename T>
   interface IEqualityComparable
```

Header

```
   #include <Generic\Interfaces\IEqualityComparable.mqh>
```

Inheritance Hierarchy

IEqualityComparable

Direct descendants

IComparable

Class Methods

Method	Description
Equals	Compares the current object with the specified value
HashCode	Calculates the hash code value for the current object

Equals

Compares the current object with the specified value.

```
bool Equals(
   T   value      // the value to compare
   );
```

Parameters

value

[in] The value to compare the current object with.

Return Value

Returns true if the objects are equal, or false otherwise.

HashCode

Calculates the hash code value for the current object.

```
int HashCode();
```

Return Value

Returns the hash code.

IComparable<T>

IComparable<T> is an interface for implementing objects that can be compared to find out whether one is greater than, less than or equal to the other one

Description

The IComparable<T> interface defines a method to compare the current object to another object of the same type, on the basis of which the collection of these objects can be sorted.

Declaration

```
template<typename T>
interface IComparable : public IEqualityComparable<T>
```

Header

```
#include <Generic\Interfaces\IComparable.mqh>
```

Inheritance Hierarchy

IEqualityComparable
    IComparable

Direct descendants

    CKeyValuePair

Class Methods

Method	Description
Compare	Compares the current object with the specified value

Compare

Compares the current object with the specified value.

```
int Compare(
   T   value      // the value to compare
   );
```

Parameters

value

[in] The value to compare the current object with.

Return Value

Returns a number that expresses the ratio of the current and passed object:

- if the result is less than zero, the current object is less than the passed one
- if the result is zero, the current object is equal to than the passed one
- if the result is greater than zero, the current object is greater than the passed one

IComparer<T>

IComparer<T> is an interface for implementing a generic class that compares two object of the T type, whether one is greater than, less than or equal to the other one

Description

The IComparer<T> interface determines a method to compare two objects of the T type, on the basis of which a collection of these objects can be sorted.

Declaration

```
   template<typename T>
   interface IComparer
```

Header

```
   #include <Generic\Interfaces\IComparer.mqh>
```

Inheritance Hierarchy

IComparer

Direct descendants

CDefaultComparer

Class Methods

Method	Description
Compare	Compares two values of type T

Compare

Compares two values of type T.

```
int Compare(
    T  x,      // the first value
    T  y       // the second value
    );
```

Parameters

x

[in] The first value to compare.

y

[in] The first value to compare.

Return Value

Returns a number that expresses the ratio of the two compared values:

- if the result is less than zero, x is less than y (x<y)
- if the result is equal to zero, x is equal to y (x=y)
- if the result is greater than zero, x is greater than y (x>y)

IEqualityComparer<T>

IEqualityComparer<T> is an interface for implementing a generic class that compares two object of the T type.

Description

The IEqualityComparer<T> interface defines methods to retrieve the hash code of a T type object and to check whether two objects of type T are equal.

Declaration

```
template<typename T>
interface IEqualityComparer
```

Header

```
#include <Generic\Interfaces\IEqualityComparer.mqh>
```

Inheritance Hierarchy

IEqualityComparer

Direct descendants

CDefaultEqualityComparer

Class Methods

Method	Description
Equals	Compares two values of type T
HashCode	Calculates the hash code value based on the T type object

Equals

Compares two values of type T.

```
bool Equals(
   T  x,      // the first value
   T  y       // the second value
   );
```

Parameters

x

[in] The first value to compare.

y

[in] The second value to compare.

Return Value

Returns true if the values are equal, or false otherwise.

HashCode

Calculates the hash code value based on the T type object.

```
int HashCode(
   T  value      // an object for calculation
   );
```

Parameters

value

[in] The object for which you want to get the hash code.

Return Value

Returns the hash code.

IList<T>

IList<T> is an interface for implementing generic data lists.

Description

The IList<T> interface defines basic methods to work with lists, such as to access an element by index, to search and delete elements, sort, and others.

Declaration

```
template<typename T>
interface IList : public ICollection<T>
```

Header

```
#include <Generic\Interfaces\IList.mqh>
```

Inheritance Hierarchy

ICollection
    IList

Direct descendants

CArrayList

Class Methods

Method	Description
TryGetValue	Gets a list element at the specified index
TrySetValue	Changes a value from the list at the specified index
Insert	Inserts an element into the list at the specified index
IndexOf	Searches for the first occurrence of a value in a list
LastIndexOf	Searches for the last occurrence of a value in a list
RemoveAt	Removes a list element at the specified index

TryGetValue

Gets a list element at the specified index.

```
bool TryGetValue(
   const int  index,      // element index
   T&         value       // a variable for writing
   );
```

Parameters

index

[in] The index of the element from the list.

&value

[out] The variable to which the specified value of the element from the list will be written.

Return Value

Returns true on successful, or false otherwise.

TrySetValue

Changes a value from the list at the specified index.

```
bool TrySetValue(
   const int  index,     // element index
   T          value      // new value
   );
```

Parameters

index

[in] The index of the element from the list.

value

[in] The new value to assign to the specified list element.

Return Value

Returns true on successful, or false otherwise.

Insert

Inserts an element into the list at the specified index.

```
bool Insert(
   const int  index,      // index to insert at
   T          item        // the value to be inserted
   );
```

Parameters

index

[in] The index to insert at.

item

[in] The value to be inserted at the specified index.

Return Value

Returns true on successful, or false otherwise.

IndexOf

Searches for the first occurrence of a value in a list.

```
int  IndexOf(
   T   item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns the index of the first found element. If the value is not found, returns -1.

LastIndexOf

Searches for the last occurrence of a value in a list.

```
int LastIndexOf(
   T  item     // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns the index of the last found element. If the value is not found, returns -1.

RemoveAt

Removes a list element at the specified index.

```
bool RemoveAt(
   const int  index      // element index
   );
```

Parameters

index

[in] The index of the element that you want to delete.

Return Value

Returns true on successful, or false otherwise.

IMap<TKey, TValue>

IMap<TKey, TValue> is an interface for implementing generic collections of key/value pairs.

Description

The IMap<TKey, TValue> interface defines basic methods to work with collections whose data are stored as key/value pairs.

Declaration

```
   template<typename TKey, typename TValue>
   interface IMap : public ICollection<TKey>
```

Header

```
   #include <Generic\Interfaces\IMap.mqh>
```

Inheritance Hierarchy

ICollection
    IMap

Direct descendants

CHashMap, CSortedMap

Class Methods

Method	Description
Add	Adds a key/value pair to a collection
Contains	Determines whether a collection contains the key/value table with the specified key
Remove	Removes the first occurrence of a key/value pair from a collection
TryGetValue	Gets an element with the specified key from a collection
TrySetValue	Changes the value of the key/value pair from a collection at the specified key
CopyTo	Copies all key/value pairs from a collection to specified arrays, starting at the specified index

Add

Adds a key/value pair to a collection.

```
bool Add(
   TKey     key,       // key
   TValue   value      // value
   );
```

Parameters

key

[in] Key.

value

[in] Value.

Return Value

Returns true on successful, or false otherwise.

Contains

Determines whether a collection contains the key/value table with the specified key.

```
bool Contains(
   TKey     key,       // key
   TValue   value      // value
   );
```

Parameters

key

[in] Key.

value

[in] Value.

Return Value

Returns true, if the collection contains the key/value pair with the specified key and value, or false otherwise.

Remove

Removes the first occurrence of a key/value pair from a collection.

```
bool  Remove(
   TKey   key      // key
   );
```

Parameters

key

[in] Key.

Return Value

Returns true on successful, or false otherwise.

TryGetValue

Gets an element with the specified key from a collection.

```
bool TryGetValue(
   TKey      key,       // key
   TValue&   value      // a variable for writing the value
   );
```

Parameters

key

[in] Key.

&value

[out] The variable to which the specified value of the key/value pair will be written.

Return Value

Returns true on successful, or false otherwise.

TrySetValue

Changes the value of the key/value pair from the collection at the specified key.

```
bool TrySetValue(
   TKey     key,        // key
   TValue   value       // new value
   );
```

Parameters

key

[in] Key.

value

[in] The new value to assign to the specified key-value pair.

Return Value

Returns true on successful, or false otherwise.

CopyTo

Copies all key/value pairs from a collection to the specified arrays, starting at the specified index.

```
int  CopyTo(
   TKey&        dst_keys[],         // an array for writing keys
   TValue&      dst_values[],       // an array for writing values
   const int    dst_start=0         // the starting index for writing
   );
```

Parameters

&dst_keys[]

[out] An array to which all keys from the collection will be written.

&dst_values[]

[out] An array to which values of corresponding keys from the collection will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied key/value pairs.

ISet<T>

ISet<T> is an interface for implementing generic data sets.

Description

The ISet interface defines basic methods to work with sets, such as: the union and intersection of sets, definition of strict and non-strict subsets, and others.

Declaration

```
template<typename T>
interface ISet : public ICollection<T>
```

Header

```
#include <Generic\Interfaces\ISet.mqh>
```

Inheritance Hierarchy

ICollection
ISet

Direct descendants

CHashSet, CSortedSet

Class Methods

Method	Description
ExceptWith	Produces the operation of difference between the current collection and a passed collection (array)
IntersectWith	Produces the operation of intersection of the current collection and a passed collection (array)
SymmetricExceptWith	Produces the operation of symmetrical difference between the current collection and a passed collection (array)
UnionWith	Produces the union of the current collection and a passed collection (array)
IsProperSubsetOf	Determines whether the current set is a proper subset of the specified collection or array
IsProperSupersetOf	Determines whether the current set is a proper superset of the specified collection or array
IsSubsetOf	Determines whether the current set is a subset of the specified collection or array
IsSupersetOf	Determines whether the current set is a superset of the specified collection or array

Method	Description
Overlaps	Determines whether the current set overlaps the specified collection or array
SetEquals	Determines whether the current set contains all elements of the specified collection or array

ExceptWith

Produces the operation of difference between the current collection and a passed collection (array). It removes from the current collection (array) all elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void ExceptWith(
   ICollection<T>*   collection     // collection
   );
```

A version for working with an array.

```
void ExceptWith(
   T&   array[]                     // array
   );
```

Parameters

**collection*

[in] A collection to be excepted from the current set.

&collection[]

[in] An array to be excepted from the current set.

Note

The result is written to the current collection (array).

IntersectWith

Produces the operation of intersection of the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void IntersectWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void IntersectWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection with which the current set will be intersected.

&collection[]

[in] An array with which the current set will be intersected.

Note

The result is written to the current collection (array).

SymmetricExceptWith

Produces the operation of symmetrical difference between the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the source object or in the specified collection (array), but not in both of them.

A version for working with the collection that implements the ICollection<T> interface.

```
void SymmetricExceptWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void SymmetricExceptWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection to produce the symmetrical difference with.

&collection[]

[in] An array to produce the symmetrical difference with.

Note

The result is written to the current collection (array).

UnionWith

Produces the union of the current collection and a passed collection (array). It adds to the current collection (array) missing elements from the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void UnionWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void UnionWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection with which the current set will be united.

&collection[]

[in] An array with which the current set will be united.

Note

The result is written to the current collection (array).

IsProperSubsetOf

Determines whether the current set is a proper subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSubsetOf(
   ICollection<T>*   collection      // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSubsetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a proper subset, or false otherwise.

IsProperSupersetOf

Determines whether the current set is a proper superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSupersetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSupersetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a proper superset, or false otherwise.

IsSubsetOf

Determines whether the current set is a subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsSubsetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsSubsetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a subset, or false otherwise.

IsSupersetOf

Determines whether the current set is a superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsSupersetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsSupersetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a superset, or false otherwise.

Overlaps

Determines whether the current set overlaps the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool Overlaps(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool Overlaps(
   T&   array[]                      // an array to compare
   );
```

Parameters

**collection*

[in] A collection to determine overlapping.

&collection[]

[in] An array to determine overlapping.

Return Value

Returns true if the current set and a collection or an array overlap, or false otherwise.

SetEquals

Determines whether the current set contains all elements of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool SetEquals(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool SetEquals(
   T&   array[]                      // an array to compare
   );
```

Parameters

**collection*

[in] A collection to compare elements.

&collection[]

[in] A collection to compare elements.

Return Value

Returns true if the current set contains all elements of the specified collection or array, or false otherwise.

CDefaultComparer<T>

CDefaultComparer<T> is a helper class that implements the IComparer<T> generic interface based on Compare global methods.

Description

The CDefaultComparer<T> class is used by default in generic data collections, unless the user implicitly uses another class implementing the IComparer<T> interface.

Declaration

```
template<typename T>
class CDefaultComparer : public IComparer<T>
```

Header

```
#include <Generic\Internal\DefaultComparer.mqh>
```

Inheritance Hierarchy

IComparer

    CDefaultComparer

Class Methods

Method	Description
Compare	Compares two values of type T

Compare

Compares two values of type T.

```
int Compare(
   T  x,      // the first value
   T  y       // the second value
   );
```

Parameters

x

[in] The first value to compare.

y

[in] The second value to compare.

Return Value

Returns a number that expresses the ratio of the two compared values:

- if the result is less than zero, x is less than y (x<y)
- if the result is equal to zero, x is equal to y (x=y)
- if the result is greater than zero, x is greater than y (x>y)

Note

The x and y values are compared based one one of he overloads of the Compare global method depending on the T type.

CDefaultEqualityComparer<T>

CDefaultEqualityComparer<T> is a helper class that implements the IEqualityComparer<T> generic interface based on Equals<T> and GetHashCode global methods.

Description

The CDefaultEqualityComparer<T> class is used by default in generic data collections, unless the user implicitly uses another class implementing the IEqualityComparer<T> interface.

Declaration

```
template<typename T>
class CDefaultEqualityComparer : public IEqualityComparer<T>
```

Header

```
#include <Generic\Internal\DefaultEqualityComparer.mqh>
```

Inheritance Hierarchy

IEqualityComparer
    CDefaultEqualityComparer

Class Methods

Method	Description
Equals	Compares two values of type T
HashCode	Calculates the hash code value based on the T type object

Equals

Compares two values of type T.

```
bool Equals(
   T  x,      // the first value
   T  y       // the second value
   );
```

Parameters

x

[in] The first value to compare.

y

[in] The second value to compare.

Return Value

Returns true if the values are equal, or false otherwise.

HashCode

Calculates the hash code value based on the T type object.

```
int HashCode(
   T  value      // an object for calculation
   );
```

Parameters

value

[in] The object for which you want to get the hash code.

Return Value

Returns the hash code.

CRedBlackTreeNode<T>

CRedBlackTreeNode<T> is a helper class used in implementing the CRedBlackTree<T> class.

Description

The CRedBlackTreeNode<T> class is a node of the CRedBlackTree<T>. Tree navigation methods are implemented in the class.

Declaration

```
template<typename T>
class CRedBlackTreeNode
```

Header

```
#include <Generic\RedBlackTree.mqh>
```

Class Methods

Method	Description
Value	Returns and sets a node value
Parent	Returns and sets a pointer to the parent node
Left	Returns and sets a pointer to the left node
Right	Returns and sets a pointer to the right node
Color	Returns and sets a node color
IsLeaf	Determines whether the specified node is a leaf
CreateEmptyNode	Creates a new black node with no parent and children, and returns a pointer to it

Value (the Get method)

Returns the node value.

```
T Value();
```

Return Value

Returns the node value.

Value (the Set method)

Sets the node value

```
void Value(
   T  value      // node value
   );
```

Parameters

value

[in] Node value.

Parent (the Get method)

Returns a pointer to the parent node.

```
CRedBlackTreeNode<T>* Parent();
```

Return Value

Returns a pointer to the parent node.

Parent (the Set method)

Sets a pointer to the parent node.

```
void Parent(
   CRedBlackTreeNode<T>*  node      // a pointer to the parent node
   );
```

Parameters

**node*

[in] A pointer to the parent node.

Left (the Get method)

Returns a pointer to the left node.

```
CRedBlackTreeNode<T>* Left();
```

Return Value

Returns a pointer to the left node.

Left (the Set method)

Sets a pointer to the left node.

```
void Left(
   CRedBlackTreeNode<T>*  node      // a pointer to the left node
   );
```

Parameters

**node*

[in] A pointer to the left node.

Right (the Get method)

Returns a pointer to the right node.

```
CRedBlackTreeNode<T>* Right();
```

Return Value

Returns a pointer to the right node.

Right (the Set method)

Sets a pointer to the right node.

```
void Right(
   CRedBlackTreeNode<T>*  node      // a pointer to the right node
   );
```

Parameters

**node*

[in] A pointer to the right node.

Color (Get method)

Returns a node color.

```
ENUM_RED_BLACK_TREE_NODE_TYPE Color();
```

Return Value

Returns a node color.

Color (Set method)

Sets the node color.

```
void Color(
   ENUM_RED_BLACK_TREE_NODE_TYPE  clr      // node color
   );
```

Parameters

clr

[in] Node color.

Note

The color of the node is set using a value from ENUM_RED_BLACK_TREE_NODE_TYPE. It can be of two types:

- RED_BLACK_TREE_NODE_RED – the red color of the node;
- RED_BLACK_TREE_NODE_BLACK – the black color of the node.

IsLeaf

Determines whether the specified node is a leaf.

```
bool IsLeaf();
```

Return Value

Returns true if the node is a leaf, or false otherwise.

CreateEmptyNode

Creates a new black node with no parent and children, and returns a pointer to it.

```
static CRedBlackTreeNode<T>* CreateEmptyNode();
```

Return Value

Returns a pointer to the new node.

CLinkedListNode<T>

CLinkedListNode<T> is a helper class used in implementing the CLinkedListNode<T> class.

Description

The CLinkedListNode<T> class is a node of the doubly linked list CLinkedListNode<T>. List navigation methods are implemented in the class.

Declaration

```
   template<typename T>
   class CLinkedListNode
```

Header

```
   #include <Generic\LinkedList.mqh>
```

Class Methods

Method	Description
List	Returns and sets a pointer to the CLinkedList<T>
Next	Returns and sets a pointer to the next node
Previous	Returns and sets a pointer to the previous node
Value	Returns and sets the node value

List (the Get method)

Returns a pointer to the CLinkedList<T>.

```
CLinkedList<T>* List();
```

Return Value

Returns a pointer to the CLinkedList<T> linked list.

List (the Set method)

Sets a pointer to the CLinkedList<T>.

```
void List(
   CLinkedList<T>*  value     // a pointer to the list
   );
```

Parameters

**value*

[in] A pointer to the linked list CLinkedList<T>.

Next (the Get method)

Returns a pointer to the next node.

```
CLinkedListNode<T>* Next();
```

Return Value

Returns a pointer to the next node.

Next (the Set method)

Sets a pointer to the next node

```
void Next(
   CLinkedListNode<T>*  value     // a pointer to the next node
   );
```

Parameters

**value*

[in] A pointer to the next node.

Previous (the Get method)

Returns a pointer to the previous node.

```
CLinkedListNode<T>* Previous();
```

Return Value

Returns a pointer to the previous node.

Previous (the Set method)

Sets a pointer to the previous node.

```
void Previous(
   CLinkedListNode<T>*  value     // a pointer to the previous node
   );
```

Parameters

**value*

[in] A pointer to the previous node.

Value (the Get method)

Returns the node value.

```
T Value();
```

Return Value

Returns the node value.

Value (the Set method)

Sets the node value

```
void Value(
   T  value      // Node value
   );
```

Parameters

value

[in] Node value.

CKeyValuePair<TKey,TValue>

The CKeyValuePair<TKey,TValue> class implements a key/value pair.

Description

The CKeyValuePair<TKey,TValue> class implements methods for working with the key and the value of the key/value pair.

Declaration

```
   template<typename TKey, typename TValue>
   class CKeyValuePair : public IComparable<CKeyValuePair<TKey,TValue>*>
```

Header

```
   #include <Generic\HashMap.mqh>
```

Inheritance Hierarchy

- IEqualityComparable
  - IComparable
    - CKeyValuePair

Class Methods

Method	Description
Key	Gets and sets the key in the key/value pair
Value	Gets and sets the value in the key/value pair
Clone	Creates a new key/value pair whose key and value are equal to the current ones
Compare	Compares the current key/value pair to the specified one
Equals	Checks whether the current key/value pair and the specified one are equal
HashCode	Calculates the hash value based on the key/value pair

Key (the Get method)

Gets the key in the key/value pair.

```
TKey Key();
```

Return Value

Returns the key.

Key (the Set method)

Sets the key in the key/value pair.

```
void Key(
   TKey  key      // key
   );
```

Parameters

key

[in] Key.

Value (the Get method)

Gets the value in the key/value pair.

```
TValue Value();
```

Return Value

Returns the value.

Value (the Set method)

Sets the value in the key/value pair.

```
void Value(
   TValue   value      // value
   );
```

Parameters

value

[in] Value

Clone

Creates a new key/value pair whose key and value are equal to the current ones.

```
TValue>* Clone();
```

Return Value

Returns a new key/value pair

Compare

Compares the current key/value pair to the specified one.

```
int Compare(
   CKeyValuePair<TKeyTValue>*  pair     // the pair to compare
   );
```

Parameters

**pair*

[in] The pair to compare.

Return Value

Returns a number that expresses the ratio of the current and passed key-value pairs:

- if the result is less than zero, the current key/value pair is less than the passed one
- if the result is zero, the current key/value pair is equal to than the passed one
- if the result is greater than zero, the current key/value pair is greater than the passed one

Note

Key/value pairs are compared based on their keys.

Equals

Checks whether the current key/value pair and the specified one are equal.

```
bool Equals(
   CKeyValuePair<TKeyTValue>*  pair     // the pair to compare
   );
```

Parameters

**pair*

[in] The pair to compare

Return Value

Returns true if the key/value pairs are equal, or false otherwise.

Note

Key/value pairs are compared based on their keys.

HashCode

Calculates the hash value based on the key/value pair.

```
int HashCode();
```

Return Value

Returns the hash code.

Note

The hash code of the key/value pair is equal to the key hash code.

CArrayList<T>

CArrayList<T> is a generic class that implements the IList<T> interface.

Description

The CArrayList<T> class is an implementation of the dynamic data list of the T type. This class provides the basic methods to work with the list, such as to access an element by index, to search and delete elements, sort, and others.

Declaration

```
template<typename T>
class CArrayList : public IList<T>
```

Header

```
#include <Generic\ArrayList.mqh>
```

Inheritance Hierarchy

- ICollection
  - IList
    - CArrayList

Class Methods

Method	Description
Capacity	Gets and sets the current capacity of a list
Count	Returns the number of elements in the list
Contains	Determines whether a list contains an element with the specified value
TrimExcess	Sets the capacity of a list to the actual number of elements
TryGetValue	Gets an element of the list at the specified index
TrySetValue	Sets the value of the list element at the specified index
Add	Adds an element to the list
AddRange	Adds a collection or an array of elements to the list
Insert	Inserts an element into the list at the specified index
InsertRange	Inserts a collection or an array of elements into the list at the specified index
CopyTo	Copies all elements of a list to the specified array starting at the specified index
BinarySearch	Searches for the specified value in an ascending-sorted list
IndexOf	Searches for the first occurrence of a value in a list

Method	Description
LastIndexOf	Searches for the last occurrence of a value in a list
Clear	Removes all elements from a collection
Remove	Removes the first occurrence of the specified element from the list
RemoveAt	Removes an element at the specified index of the list
RemoveRange	Removes a range of elements from the list
Reverse	Reverses the order of elements in the list
Sort	Sorts elements in the list

Capacity (the Get method)

Returns the current list capacity.

```
int Capacity();
```

Return Value

Returns the current list capacity.

Capacity (the Set method)

Sets the current capacity of a list.

```
void Capacity(
   const int  capacity      // capacity value
   );
```

Parameters

capacity

[in] A new value of capacity.

Count

Returns the number of elements in the list.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether the list contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the list, or false otherwise.

TrimExcess

Sets the capacity of a list to the actual number of elements, and thus frees up unused memory.

```
void TrimExcess();
```

TryGetValue

Gets an element of the list at the specified index.

```
bool TryGetValue(
   const int  index,      // index
   T&         value       // a variable to write
   );
```

Parameters

index

[in] The index of the list element the value of which you want to get.

&value

[out] The variable to write the element value to.

Return Value

Returns true on successful, or false otherwise.

TrySetValue

Sets the value of the list element at the specified index.

```
bool TrySetValue(
   const int  index,      // index
   T          value       // element value
   );
```

Parameters

index

[in] The index of the list element the value of which you want to set.

value

[in] Sets the value of the list element.

Return Value

Returns true on successful, or false otherwise.

Add

Adds an element to the list.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

AddRange

Adds a collection or an array of elements to the list.

The version that adds an array.

```
bool AddRange(
   const T&  array[]              // an array to be added
   );
```

The version that adds a collection.

```
bool AddRange(
   ICollection<T>*  collection     // a collection to be added
   );
```

Parameters

&array[]

[in] An array to be added.

**collection*

[in] A collection to be added.

Return Value

Returns true on successful, or false otherwise.

Insert

Inserts an element into the list at the specified index.

```
bool Insert(
   const int  index,      // index to insert at
   T          item        // the value to be inserted
   );
```

Parameters

index

[in] The index to insert at.

item

[in] The value to be inserted at the specified index.

Return Value

Returns true on successful, or false otherwise.

InsertRange

Inserts a collection or an array of elements into the list at the specified index.

The version that inserts an array.

```
bool InsertRange(
   const int   index,                  // index to insert at
   const T&    array[]                 // an array to be inserted
   );
```

The version that inserts a collection.

```
bool InsertRange(
   const int          index,           // index to insert at
   ICollection<T>*    collection       // a collection to be inserted
   );
```

Parameters

index

[in] The index to insert at.

&array[]

[in] An array to be inserted at the specified index.

**collection*

[in] A collection to be inserted at the specified index.

Return Value

Returns true on successful, or false otherwise.

CopyTo

Copies all elements of a list to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // the starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the list will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

BinarySearch

Searches for the specified value in an ascending-sorted list.

The version that searches in the specified range of values using the class that implements the IComparable<T> interface for comparing elements.

```
int BinarySearch(
   const int       index,        // the starting index
   const int       count,        // the search range
   T               item,         // the search value
   IComparer<T>*   comparer      // interface to compare
   );
```

The version that searches using the class that implements the IComparable<T> interface for comparing elements.

```
int BinarySearch(
   T               item,         // the search value
   IComparer<T>*   comparer      // interface to compare
   );
```

The version that searches using the ::Compare global method for comparing elements.

```
int BinarySearch(
   T  item                       // the search value
   );
```

Parameters

index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

item

[in] The searched value.

**comparer*

[in] An interface for comparing elements.

Return Value

Returns the index of the found element. If the search value is not found, it returns the index of the smallest element, which is closest in value.

IndexOf

Searches for the first occurrence of a value in a list.

Version that searches in the entire list.

```
int IndexOf(
   T  item                          // the search value
   );
```

Version that searches from the specified position and to the end of the list.

```
int IndexOf(
   T           item,                // the search value
   const int   start_index          // the starting index
   );
```

Version that searches from the specified position in the specified range.

```
int IndexOf(
   T           item,                // the search value
   const int   start_index,         // the starting index
   const int   count                // the search range
   );
```

Parameters

item

[in] The searched value.

start_index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

Return Value

Returns the index of the first found element. If the value is not found, returns -1.

LastIndexOf

Searches for the last occurrence of a value in a list.

Version that searches in the entire list.

```
int LastIndexOf(
   T  item                        // the search value
   );
```

Version that searches from the specified position and to the end of the list.

```
int LastIndexOf(
   T          item,               // the search value
   const int  start_index         // the starting index
   );
```

Version that searches from the specified position in the specified range.

```
int LastIndexOf(
   T          item,               // the search value
   const int  start_index,        // the starting index
   const int  count               // the search range
   );
```

Parameters

item

[in] The searched value.

start_index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

Return Value

Returns the index of the last found element. If the value is not found, returns -1.

Clear

Removes all elements of a collection.

```
void Clear();
```

Remove

Removes the first occurrence of the specified element from the list.

```
bool Remove(
   T  item      // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

RemoveAt

Removes an element at the specified index of the list.

```
bool RemoveAt(
    const int   index      // index
    );
```

Parameters

index

[in] The index of the element to remove.

Return Value

Returns true on successful, or false otherwise.

RemoveRange

Removes a range of elements from the list.

```
bool RemoveRange(
   const int  start_index,      // the starting index
   const int  count             // the number of elements
   );
```

Parameters

start_index

[in] The starting index from which the deletion begins.

count

[in] The number of elements to be deleted.

Return Value

Returns true on successful, or false otherwise.

Reverse

Reverses the order of elements in the list.

The version for working with the entire list.

```
bool Reverse();
```

The version for working with the specified range of list elements.

```
bool Reverse(
   const int   start_index,      // the starting index
   const int   count             // the number of elements
   );
```

Parameters

start_index

[in] The starting index.

count

[in] The number of list elements participating in the operation.

Return Value

Returns true on successful, or false otherwise.

Sort

Sorts elements in the list.

The version that sorts all elements in the list.

```
bool Sort();
```

The version that sorts all elements in the list using the class that implements the IComparable<T> interface for comparing elements.

```
bool Sort(
   IComparer<T>*   comparer          // interface for comparing
   );
```

The version that sorts the specified range of elements in the list using the class that implements the IComparable<T> interface for comparing elements.

```
bool Sort(
   const int   start_index,      // the starting index
   const int   count             // the number of elements
   IComparer<T>*   comparer      // interface to compare
   );
```

Parameters

**comparer*

[in] An interface for comparing elements.

start_index

[in] The starting index from which sorting begins.

count

[in] The length of the sorting range.

Return Value

Returns true on successful, or false otherwise.

CHashMap<TKey, TValue>

CHashMap<TKey, TValue> is a generic class that implements the IMap<TKey, TValue> interface.

Description

The CHashMap<TKey, TValue> class is an implementation of the dynamic hash table, the data of which are stored in the form of unordered key/value pairs taking into account the key uniqueness requirement. This class provides basic methods to work with a hash table, such as to access a value by key, to search and delete a key/value pair, and others.

Declaration

```
template<typename TKey, typename TValue>
class CHashMap : public IMap<TKey, TValue>
```

Header

```
#include <Generic\HashMap.mqh>
```

Inheritance Hierarchy

- ICollection
  - IMap
    - CHashMap

Class Methods

Method	Description
Add	Adds a key/value pair to the hash table
Count	Returns the number of elements in the hash table
Comparer	Returns a pointer to the IEqualityComparer<T> interface, used to organize a hash table
Contains	Determines whether the hash table contains the specified key/value pair
ContainsKey	Determines whether the hash table contains the key/value pair with the specified key
ContainsValue	CHashMap<TKey, TValue> is a generic class that implements the IMap<TKey, TValue> interface
CopyTo	Copies all key/value pairs from the hash table to the specified arrays, starting at the specified index
Clear	Removes all elements from the hash table
Remove	Removes the first occurrence of the key/value pair from the hash table
TryGetValue	Gets an element with the specified key from the hash table

Method	Description
TrySetValue	Changes the value of a key/value pair from the hash table at the specified key

Add

Adds a key/value pair to the hash table.

A version that adds a generated key/value pair.

```
bool Add(
   CKeyValuePair<TKeyTValue>*  pair     // the key/value pair
   );
```

A version that adds a new key/value pair with the specified key and value.

```
bool Add(
   TKey     key,                          // key
   TValue   value                         // value
   );
```

Parameters

**pair*

[in] The key/value pair.

key

[in] Key.

value

[in] Value.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the hash table.

```
int Count();
```

Comparer

Returns a pointer to the IEqualityComparer<T> interface, used to organize a hash table.

```
IEqualityComparer<TKey>* Comparer() const;
```

Return Value

Returns a pointer to the IEqualityComparer<T> interface.

Contains

Determines whether the hash table contains the specified key/value pair.

The version for working with a generated key/value pair.

```
bool Contains(
   CKeyValuePair<TKeyTValue>*  item      // the key/value pair
   );
```

The version for working with a key/value pair in the form of a separately set key and value.

```
bool Contains(
   TKey    key,                          // key
   TValue  value                         // value
   );
```

Parameters

**item*

[in] The key/value pair.

key

[in] Key.

value

[in] Value.

Return Value

Returns true, if the hash table contains the key/value pair with the specified key and value, or false otherwise.

ContainsKey

Determines whether the hash table contains the key/value pair with the specified key.

```
bool ContainsKey(
   TKey  key      // key
   );
```

Parameters

key

[in] Key.

Return Value

Returns true if the hash table contains a key/value pair with the specified key, or false otherwise.

ContainsValue

Determines whether the hash table contains the key/value pair with the specified value.

```
bool ContainsValue(
   TValue   value      // value
   );
```

Parameters

value

[in] Value.

Return Value

Returns true if the hash table contains a key/value pair with the specified value, or false otherwise.

CopyTo

Copies all key/value pairs from the hash table to the specified arrays, starting at the specified index.

The version that copies a hash table to the array of key/value pairs.

```
int CopyTo(
   CKeyValuePair<TKeyTValue>*&  dst_array[],      // an array for writing key/value pa
   const int                    dst_start=0       // the starting index for writing
   );
```

The version that copies a hash table to separate arrays for keys and values.

```
int CopyTo(
   TKey&      dst_keys[],                          // an array for writing keys
   TValue&    dst_values[],                        // an array for writing values
   const int  dst_start=0                          // starting index for writing
   );
```

Parameters

*&dst_array[]

[out] An array to which all pairs from the hash table will be written.

&dst_keys[]

[out] An array to which all keys from the hash table will be written.

&dst_values[]

[out] An array to which all values from the hash table will be written.

dst_start=0

[in] The array index from which copying starts.

Return Value

Returns the number of copied key/value pairs.

Clear

Removes all elements from the hash table.

```
void Clear();
```

Remove

Removes the first occurrence of the key/value pair from the hash table.

The version that removes a key-value pair based on the generated key-value pair.

```
bool Remove(
   CKeyValuePair<TKeyTValue>*  item     // the key/value pair"
   );
```

The version that removes a key-value pair based on the key.

```
bool Remove(
   TKey  key                            // key
   );
```

Parameters

**item*

[in] The key/value pair.

key

[in] Key.

Return Value

Returns true on successful, or false otherwise.

TryGetValue

Gets an element with the specified key from the hash table.

```
bool TryGetValue(
   TKey      key,       // key
   TValue&   value      // a variable for writing the value
   );
```

Parameters

key

[in] Key.

&value

[out] The variable to which the specified value of the key/value pair will be written.

Return Value

Returns true on successful, or false otherwise.

TrySetValue

Changes the value of a key/value pair from the hash table at the specified key.

```
bool TrySetValue(
   TKey     key,       // key
   TValue   value      // new value
   );
```

Parameters

key

[in] Key.

value

[in] The new value to assign to the specified key-value pair.

Return Value

Returns true on successful, or false otherwise.

CHashSet<T>

CHashSet<T> is a generic class that implements the ISet<T> interface.

Description

The CHashSet<T> class is an implementation of the unordered dynamic data set of type T, with the required uniqueness of each value. This class provides basic methods to work with sets and related operations, such as: the union and intersection of sets, definition of strict and non-strict subsets, and others.

Declaration

```
   template<typename T>
   class CHashSet : public ISet<T>
```

Header

```
   #include <Generic\HashSet.mqh>
```

Inheritance Hierarchy

ICollection
  ISet
    CHashSet

Class Methods

Method	Description
Add	Adds an element to a set
Count	Returns the number of elements in a set
Comparer	Determines whether a set contains an element with the specified value
Contains	Returns a pointer to the IEqualityComparer<T> interface, used to organize a set
TrimExcess	Sets the capacity of a set to the actual number of elements, and thus frees up unused memory
CopyTo	Copies all elements of a set to the specified array starting at the specified index
Clear	Removes all elements from a set
Remove	Removes the specified element from a set
ExceptWith	Produces the operation of difference between the current collection and a passed collection (array)
IntersectWith	Produces the operation of intersection of the current collection and a passed collection (array)

Method	Description
SymmetricExceptWith	Produces the operation of symmetrical difference between the current collection and a passed collection (array)
UnionWith	Produces the union of the current collection and a passed collection (array)
IsProperSubsetOf	Determines whether the current set is a proper subset of the specified collection or array
IsProperSupersetOf	Determines whether the current set is a proper superset of the specified collection or array
IsSubsetOf	Determines whether the current set is a subset of the specified collection or array
IsSupersetOf	Determines whether the current set is a superset of the specified collection or array
Overlaps	Determines whether the current set overlaps the specified collection or array
SetEquals	Determines whether the current set contains all elements of the specified collection or array

Add

Adds an element to a set.

```
bool  Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in a set.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether a set contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the set, or false otherwise.

Comparer

Returns a pointer to the IEqualityComparer<T> interface, used to organize a set.

```
IEqualityComparer<T>* Comparer() const;
```

Return Value

Returns a pointer to the IEqualityComparer<T> interface.

TrimExcess

Sets the capacity of a set to the actual number of elements, and thus frees up unused memory.

```
void TrimExcess();
```

CopyTo

Copies all elements of a set to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the set will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements from a set.

```
void Clear();
```

Remove

Removes the specified element from a set.

```
bool  Remove(
   T  item        // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

ExceptWith

Produces the operation of difference between the current collection and a passed collection (array). It removes from the current collection (array) all elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void ExceptWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void ExceptWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection to be excepted from the current set.

&collection[]

[in] An array to be excepted from the current set.

Note

The result is written to the current collection (array).

IntersectWith

Produces the operation of intersection of the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void IntersectWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void IntersectWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection with which the current set will be intersected.

&collection[]

[in] An array with which the current set will be intersected.

Note

The result is written to the current collection (array).

SymmetricExceptWith

Produces the operation of symmetrical difference between the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the source object or in the specified collection (array), but not in both of them.

A version for working with the collection that implements the ICollection<T> interface.

```
void SymmetricExceptWith(
   ICollection<T>*   collection     // collection
   );
```

A version for working with an array.

```
void SymmetricExceptWith(
   T&   array[]                     // array
   );
```

Parameters

**collection*

[in] A collection to produce the symmetrical difference with.

&collection[]

[in] An array to produce the symmetrical difference with.

Note

The result is written to the current collection (array).

UnionWith

Produces the union of the current collection and a passed collection (array). It adds to the current collection (array) missing elements from the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void UnionWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void UnionWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection with which the current set will be united.

&collection[]

[in] An array with which the current set will be united.

Note

The result is written to the current collection (array).

IsProperSubsetOf

Determines whether the current set is a proper subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSubsetOf(
   ICollection<T>*   collection      // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSubsetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a proper subset, or false otherwise.

IsProperSupersetOf

Determines whether the current set is a proper superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSupersetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSupersetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a proper superset, or false otherwise.

IsSubsetOf

Determines whether the current set is a subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsSubsetOf(
   ICollection<T>*   collection      // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsSubsetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a subset, or false otherwise.

IsSupersetOf

Determines whether the current set is a superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsSupersetOf(
   ICollection<T>*   collection      // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsSupersetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current set is a superset, or false otherwise.

Overlaps

Determines whether the current set overlaps the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool Overlaps(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool Overlaps(
   T&   array[]                      // an array to compare
   );
```

Parameters

**collection*

[in] A collection to determine overlapping.

&collection[]

[in] An array to determine overlapping.

Return Value

Returns true if the current set and a collection or an array overlap, or false otherwise.

SetEquals

Determines whether the current set contains all elements of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool SetEquals(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool SetEquals(
   T&   array[]                      // an array to compare
   );
```

Parameters

**collection*

[in] A collection to compare elements.

&collection[]

[in] An array to compare elements.

Return Value

Returns true if the current set contains all elements of the specified collection or array, or false otherwise.

CLinkedList<T>

CLinkedList<T> is a generic class that implements the ICollection<T> interface.

Description

The CLinkedList<T> class is an implementation of the dynamic doubly linked data list of the T type. This class provides basic methods to work with doubly linked lists, such as to add, delete, search elements, and others.

Declaration

```
   template<typename T>
   class CLinkedList : public ICollection<T>
```

Header

```
   #include <Generic\LinkedList.mqh>
```

Inheritance Hierarchy

ICollection

CLinkedList

Class Methods

Method	Description
Add	Adds an element to a linked list
AddAfter	Adds an element after the specified node in the linked list
AddBefore	Adds an element before the specified node in the linked list
AddFirst	Adds an element at the beginning of the linked list
AddLast	Adds an element at the end of the linked list
Count	Returns the number of elements in the linked list
Head	Returns a pointer to the first node of the linked list
First	Returns a pointer to the first node of the linked list
Last	Returns a pointer to the last node of the linked list
Contains	Determines whether the linked list contains an element with the specified value
CopyTo	Copies all elements of the linked list to the specified array starting at the specified index
Clear	Removes all elements from a linked list
Remove	Removes the first occurrence of the specified element from the linked list

Method	Description
RemoveFirst	Removes the first element of the linked list
RemoveLast	Removes the last element of the linked list
Find	Searches for the first occurrence of the specified value in the linked list
FindLast	Searches for the last occurrence of the specified value in the linked list

Add

Adds an element to a linked list.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

AddAfter

Adds an element after the specified node in the linked list.

The version that adds an element by value.

```
CLinkedListNode<T>* AddAfter(
   CLinkedListNode<T>*   node,        // the node after which the element should be ad
   T                     value        // the element to add
   );
```

Return Value

Returns a pointer to the added node.

The version that adds an element as a formed node by value.

```
bool AddAfter(
   CLinkedListNode<T>*   node,        // the node after which the element should be ad
   CLinkedListNode<T>*   new_node     // the node to be added
   );
```

Parameters

**node*

[in] The node of the linked list, after which a new element will be added.

value

[in] An element to be added.

**new_node*

[in] A node to be added.

Return Value

Returns true on successful, or false otherwise.

AddBefore

Adds an element before the specified node in the linked list.

The version that adds an element by value.

```
CLinkedListNode<T>* AddBefore(
   CLinkedListNode<T>*  node,          // the node before which the element should be ad
   T                    value          // the element to add
   );
```

Return Value

Returns a pointer to the added node.

The version that adds an element as a formed node by value.

```
bool AddBefore(
   CLinkedListNode<T>*  node,          // the node before which the element should be ad
   CLinkedListNode<T>*  new_node       // the node to be added
   );
```

Parameters

**node*

[in] The node of the linked list, before which a new element will be added.

value

[in] An element to be added.

**new_node*

[in] A node to be added.

Return Value

Returns true on successful, or false otherwise.

AddFirst

Adds an element at the beginning of the linked list.

The version that adds an element by value.

```
CLinkedListNode<T>* AddFirst(
   T   value                          // an element to add
   );
```

Return Value

Returns a pointer to the added node.

The version that adds an element as a formed node by value.

```
bool AddFirst(
   CLinkedListNode<T>*   node      // the node to add
   );
```

Parameters

value

[in] An element to be added.

**node*

[in] A node to be added.

Return Value

Returns true on successful, or false otherwise.

AddLast

Adds an element at the end of the linked list

The version that adds an element by value.

```
CLinkedListNode<T>* AddLast(
   T  value                      // an element to add
   );
```

Return Value

Returns a pointer to the added node.

The version that adds an element as a formed node by value.

```
bool AddLast(
   CLinkedListNode<T>*  node     // the node to add
   );
```

Parameters

value

[in] An element to be added.

**node*

[in] A node to be added.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the linked list.

```
int Count();
```

Return Value

Returns the number of elements.

Head

Returns a pointer to the first node of the linked list.

```
CLinkedListNode<T>* Head();
```

Return Value

Returns a pointer to the first node.

First

Returns a pointer to the first node of the linked list.

```
CLinkedListNode<T>* First();
```

Return Value

Returns a pointer to the first node.

Last

Returns a pointer to the last node of the linked list.

```
CLinkedListNode<T>* Last();
```

Return Value

Returns a pointer to the last node.

Contains

Determines whether the linked list contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the linked list, or false otherwise.

CopyTo

Copies all elements of the linked list to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // the starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the linked list will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements of a collection.

```
void Clear();
```

Remove

Removes the first occurrence of the specified element from the linked list.

The version that removes an element by value.

```
bool Remove(
   T  item                               // the element value
   );
```

The version that removes an element by a pointer to a node.

```
bool Remove(
   CLinkedListNode<T>*  node      // the element node
   );
```

Parameters

item

[in] The value of the element to be deleted.

**node*

[in] The node of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

RemoveFirst

Removes the first element of the linked list.

```
bool RemoveFirst();
```

Return Value

Returns true on successful, or false otherwise.

RemoveLast

Removes the last element of the linked list.

```
bool RemoveLast();
```

Return Value

Returns true on successful, or false otherwise.

Find

Searches for the first occurrence of the specified value in the linked list.

```
CLinkedListNode<T>* Find(
   T  value      // the search value
   );
```

Parameters

value

[in] The searched value.

Return Value

Returns a pointer to the first found node containing the search value on success, or NULL otherwise.

FindLast

Searches for the last occurrence of the specified value in the linked list.

```
CLinkedListNode<T>* FindLast(
   T   value      // the search value
   );
```

Parameters

value

[in] The searched value.

Return Value

Returns a pointer to the last found node containing the search value on success, or NULL otherwise.

CQueue<T>

CQueue<T> is a generic class that implements the ICollection<T> interface.

Description

The CQueue<T> class is a dynamic collection of T type data, which is organized as a queue that operates on the FIFO (first in, first out) principle.

Declaration

```
   template<typename T>
   class CQueue : public ICollection<T>
```

Header

```
   #include <Generic\Queue.mqh>
```

Inheritance Hierarchy

ICollection
    CQueue

Class Methods

Method	Description
Add	Adds an element to a queue
Enqueue	Adds an element to a queue
Count	Returns the number of elements in the queue
Contains	Determines whether the queue contains an element with the specified value
TrimExcess	Sets the capacity of a queue to the actual number of elements, and thus frees up unused memory
CopyTo	Copies all elements of a queue to the specified array starting at the specified index
Clear	Removes all elements from a queue
Remove	Removes the first occurrence of the specified element from the queue
Dequeue	Returns the starting element and removes it from the queue
Peek	Returns the starting element without removing it from the queue

Add

Adds an element to a queue.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

Enqueue

Adds an element to a queue.

```
bool Enqueue(
   T  value      // element to add
   );
```

Parameters

value

[in] An element to be added.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the queue.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether the queue contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the queue, or false otherwise.

TrimExcess

Sets the capacity of a queue to the actual number of elements, and thus frees up unused memory.

```
void TrimExcess();
```

CopyTo

Copies all elements of a queue to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // the starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the queue will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements from a queue.

```
void Clear();
```

Remove

Removes the first occurrence of the specified element from the queue.

```
bool Remove(
   T  item      // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

Dequeue

Returns the first element and removes it from the queue.

```
T Dequeue();
```

Return Value

Returns the starting element.

Peek

Returns the first element not removing it from the queue.

```
T Peek();
```

Return Value

Returns the starting element.

CRedBlackTree<T>

CRedBlackTree<T> is a generic class that implements the ICollection<T> interface.

Description

The CRedBlackTree<T> class is an implementation of a dynamic red-black tree whose nodes store T type data. The class provides basic methods to work with red-black trees, such as to add, delete, search for the maximum and minimum value, and more.

Declaration

```
   template<typename T>
   class CRedBlackTree : public ICollection<T>
```

Header

```
   #include <Generic\RedBlackTree.mqh>
```

Inheritance Hierarchy

ICollection
    CRedBlackTree

Class Methods

Method	Description
Add	Adds an element to a red-black tree
Root	Returns a pointer to the root of the red-black tree
Count	Returns the number of elements in the red-black tree
Contains	Determines whether the red-black tree contains an element with the specified value
Comparer	Returns a pointer to the IComparer<T> interface used to organize a red-black tree
TryGetMin	Gets the minimum element of a red-black tree
TryGetMax	Gets the maximum element of a red-black tree
CopyTo	Copies all elements of a red-black tree to the specified array starting at the specified index
Clear	Removes all elements from a red-black tree
Remove	Removes the occurrence of the specified element from a red-black tree
RemoveMin	Removes an element with the minimum value from a red-black tree
RemoveMax	Removes an element with the maximum value from a red-black tree
Find	Searches for the occurrence of a specified value in a red-black tree

Method	Description
FindMax	Searches for an element with the maximum value in a red-black tree
FindMin	Searches for an element with the minimum value in a red-black tree

Add

Adds an element to a red-black tree.

```
bool Add(
   T  value      // element to add
   );
```

Parameters

value

[in] An element to be added.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the red-black tree.

```
int Count();
```

Return Value

Returns the number of elements.

Root

Returns a pointer to the root of the red-black tree.

```
CRedBlackTreeNode<T>* Root();
```

Return Value

Returns a pointer to the root.

Contains

Determines whether the red-black tree contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the red-black tree, or false otherwise.

Comparer

Returns a pointer to the IComparer<T> interface used to organize a red-black tree.

```
IComparer<T>* Comparer() const;
```

Return Value

Returns a pointer to the IComparer<T> interface.

TryGetMin

Gets the minimum element of a red-black tree.

```
bool TryGetMin(
   T&   min      // a variable for writing the value
   );
```

Parameters

&min

[out] The variable to which the minimum value will be written.

Return Value

Returns true on successful, or false otherwise.

TryGetMax

Gets the maximum element of a red-black tree.

```
bool TryGetMax(
   T&   max      // a variable for writing the value
   );
```

Parameters

&max

[out] The variable to which the maximum value will be written.

Return Value

Returns true on successful, or false otherwise.

CopyTo

Copies all elements of a red-black tree to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the red-black tree will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements from a red-black tree.

```
void Clear();
```

Remove

Removes the occurrence of the specified element from a red-black tree.

The version that removes an element with the specified value.

```
bool Remove(
   T   value                              // the element value
   );
```

The version that removes an element by a pointer to a node.

```
bool Remove(
   CRedBlackTreeNode<T>*  node      // the element node
   );
```

Parameters

item

[in] The value of the element to be deleted.

**node*

[in] The node of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

RemoveMin

Removes an element with the minimum value from a red-black tree.

```
bool RemoveMin();
```

Return Value

Returns true on successful, or false otherwise.

RemoveMax

Removes an element with the maximum value from a red-black tree.

```
bool RemoveMax();
```

Return Value

Returns true on successful, or false otherwise.

Find

Searches for the occurrence of a specified value in a red-black tree.

```
CRedBlackTreeNode<T>* Find(
   T  value      // the search value
   );
```

Parameters

value

[in] The searched value.

Return Value

Returns a pointer to the found node containing the search value on success, or NULL otherwise.

FindMin

Searches for an element with the minimum value in a red-black tree.

```
CRedBlackTreeNode<T>* FindMin();
```

Return Value

Returns a pointer to the node containing the minimum value on success, or NULL otherwise.

FindMax

Searches for an element with the maximum value in a red-black tree.

```
CRedBlackTreeNode<T>* FindMax();
```

Return Value

Returns a pointer to the node containing the maximum value on success, or NULL otherwise.

CSortedMap<TKey,TValue>

CSortedMap<TKey,TValue> is a generic class that implements the IMap<TKey,TValue> interface.

Description

The CSortedMap<TKey,TValue> class is an implementation of a dynamic hash table whose data are stored as key/value pairs sorted by key and taking into account the key uniqueness requirement. This class provides basic methods to work with a hash table, such as to access a value by key, to search and delete a key/value pair, and others.

Declaration

```
   template<typename TKey, typename TValue>
   class CSortedMap : public IMap<TKey, TValue>
```

Header

```
   #include <Generic\SortedMap.mqh>
```

Inheritance Hierarchy

- ICollection
  - IMap
    - CSortedMap

Class Methods

Method	Description
Add	Adds a key/value pair to the hash table
Count	Returns the number of elements in the sorted hash table
Contains	Determines whether the sorted hash table contains the specified key/value table
ContainsKey	Determines whether the sorted hash table contains the key/value table with the specified key
ContainsValue	Determines whether the sorted hash table contains the key/value table with the specified value
Comparer	Returns a pointer to the IComparer<T> interface, used to organize a sorted hash table
CopyTo	Copies all key/value pairs from the sorted hash table to the specified arrays, starting at the specified index
Clear	Removes all elements from the sorted hash table
Remove	Removes the first occurrence of the key/value pair from the sorted hash table

Method	Description
TryGetValue	Gets an element with the specified key from the sorted hash table
TrySetValue	Changes a key/value pair with the specified key from the sorted hash table

Add

Adds a key/value pair to the hash table.

A version that adds a generated key/value pair.

```
bool Add(
   CKeyValuePair<TKeyTValue>*  pair      // the key/value pair
   );
```

A version that adds a new key/value pair with the specified key and value.

```
bool Add(
   TKey     key,                         // key
   TValue   value                        // value
   );
```

Parameters

**pair*

[in] The key/value pair.

key

[in] Key.

value

[in] Value.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the sorted hash table.

```
int Count();
```

Comparer

Returns a pointer to the IComparer<T> interface, used to organize a sorted hash table.

```
IComparer<TKey>* Comparer() const;
```

Return Value

Returns a pointer to the IComparer<T> interface.

Contains

Determines whether the sorted hash table contains the specified key/value table.

The version for working with a generated key/value pair.

```
bool Contains(
   CKeyValuePair<TKeyTValue>*  item     // the key/value pair
   );
```

The version for working with a key/value pair in the form of a separately set key and value.

```
bool Contains(
   TKey     key,                        // key
   TValue   value                       // value
   );
```

Parameters

**item*

[in] The key/value pair.

key

[in] Key.

value

[in] Value.

Return Value

Returns true, if the sorted hash table contains the key/value pair with the specified key and value, or false otherwise.

ContainsKey

Determines whether the sorted hash table contains the key/value table with the specified key.

```
bool ContainsKey(
   TKey  key      // key
   );
```

Parameters

key

[in] Key.

Return Value

Returns true, if the sorted hash table contains the key/value pair with the specified key, or false otherwise.

ContainsValue

Determines whether the sorted hash table contains the key/value table with the specified value.

```
bool ContainsValue(
   TValue   value      // value
   );
```

Parameters

value

[in] Value.

Return Value

Returns true, if the sorted hash table contains the key/value pair with the specified value, or false otherwise.

CopyTo

Copies all key/value pairs from the sorted hash table to the specified arrays, starting at the specified index.

The version that copies a hash table to the array of key/value pairs.

```
int CopyTo(
   CKeyValuePair<TKeyTValue>*&  dst_array[],     // an array for writing key/value pai
   const int                    dst_start=0      // the starting index for writing
   );
```

The version that copies a hash table to separate arrays for keys and values.

```
int CopyTo(
   TKey&      dst_keys[],                        // an array for writing keys
   TValue&    dst_values[],                      // an array for writing values
   const int  dst_start=0                        // the starting index for writing
   );
```

Parameters

**&dst_array[]*

[out] An array to which all pairs from the hash table will be written.

&dst_keys[]

[out] An array to which all keys from the hash table will be written.

&dst_values[]

[out] An array to which all values from the hash table will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied key/value pairs.

Clear

Removes all elements from the sorted hash table.

```
void Clear();
```

Remove

Removes the first occurrence of the key/value pair from the sorted hash table.

The version that removes a key-value pair based on the generated key-value pair.

```
bool Remove(
   CKeyValuePair<TKeyTValue>*  item      // the key/value pair
   );
```

The version that removes a key-value pair based on the key.

```
bool Remove(
   TKey  key                             // key
   );
```

Parameters

**item*

[in] The key/value pair.

key

[in] Key.

Return Value

Returns true on successful, or false otherwise.

TryGetValue

Gets an element with the specified key from the sorted hash table.

```
bool TryGetValue(
   TKey      key,       // key
   TValue&   value      // a variable for writing the value
   );
```

Parameters

key

[in] Key.

&value

[out] The variable to which the specified value of the key/value pair will be written.

Return Value

Returns true on successful, or false otherwise.

TrySetValue

Changes the value of the key/value pair from the sorted hash table at the specified key.

```
bool TrySetValue(
   TKey     key,        // key
   TValue   value       // new value
   );
```

Parameters

key

[in] Key.

value

[in] The new value to assign to the specified key-value pair.

Return Value

Returns true on successful, or false otherwise.

CSortedSet<T>

CSortedSet<T> is a generic class that implements the ISet<T> interface.

Description

The CSortedSet<T> class is an implementation of the sorted dynamic data set of type T, with the required uniqueness of each value. This class provides basic methods to work with sets and related operations, such as: the union and intersection of sets, definition of strict and non-strict subsets, and others.

Declaration

```
template<typename T>
class CSortedSet : public ISet<T>
```

Header

```
#include <Generic\SortedSet.mqh>
```

Inheritance Hierarchy

- ICollection
  - ISet
    - CSortedSet

Class Methods

Method	Description
Add	Adds an element to a sorted set
Count	Returns the number of elements in a sorted set
Contains	Determines whether a sorted set contains an element with the specified value
Comparer	Returns a pointer to the IComparer<T> interface, used to organize a sorted set
TryGetMin	Gets the minimum element from a sorted set
TryGetMax	Gets the maximum element from a sorted set
CopyTo	Copies all elements of a sorted set to the specified array starting at the specified index
Clear	Removes all elements from a sorted set
Remove	Removes the occurrence of the specified element from a sorted set
ExceptWith	Produces the operation of difference between the current collection and a passed collection (array)

Method	Description
IntersectWith	Produces the operation of intersection of the current collection and a passed collection (array)
SymmetricExceptWith	Produces the operation of symmetrical difference between the current collection and a passed collection (array)
UnionWith	Produces the union of the current collection and a passed collection (array)
IsProperSubsetOf	Determines whether the current sorted set is a proper subset of the specified collection or array
IsProperSupersetOf	Determines whether the current sorted set is a proper superset of the specified collection or array
IsSubsetOf	Determines whether the current sorted set is a subset of the specified collection or array
IsSupersetOf	Determines whether the current sorted set is a superset of the specified collection or array
Overlaps	Determines whether the current sorted set overlaps the specified collection or array
SetEquals	Determines whether the current sorted set contains all elements of the specified collection or array
GetViewBetween	Gets from the current sorted set a subset specified by the minimum and maximum values
GetReverse	Gets a copy of the current sorted set, in which all the elements are arranged in a reverse order

Add

Adds an element to a sorted set.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the sorted set.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether the sorted set contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the set, or false otherwise.

Comparer

Returns a pointer to the IComparer<T> interface, used to organize a sorted set.

```
IComparer<T>* Comparer() const;
```

Return Value

Returns a pointer to the IComparer<T> interface.

TryGetMin

Gets the minimum element from the sorted set.

```
bool TryGetMin(
   T&   min       // a variable for writing the value
   );
```

Parameters

&min

[out] The variable to which the minimum value will be written.

Return Value

Returns true on successful, or false otherwise.

TryGetMax

Gets the maximum element from the sorted set.

```
bool TryGetMax(
   T&   max      // a variable for writing the value
   );
```

Parameters

&max

[out] The variable to which the maximum value will be written.

Return Value

Returns true on successful, or false otherwise.

CopyTo

Copies all elements of a sorted set to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // the starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the set will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements from a sorted set.

```
void Clear();
```

Remove

Removes the occurrence of the specified element from the sorted set.

```
bool Remove(
   T  item      // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

ExceptWith

Produces the operation of difference between the current collection and a passed collection (array). It removes from the current collection (array) all elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void ExceptWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void ExceptWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection to be excepted from the current sorted set.

&collection[]

[in] An array to be excepted from the current sorted set.

Note

The result is written to the current collection (array).

IntersectWith

Produces the operation of intersection of the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void IntersectWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void IntersectWith(
   T&   array[]                      // array
   );
```

Parameters

*collection

[in] A collection with which the current set will be intersected.

&collection[]

[in] An array with which the current set will be intersected.

Note

The result is written to the current collection (array).

SymmetricExceptWith

Produces the operation of symmetrical difference between the current collection and a passed collection (array). It modifies the current collection to only contain elements that are present in the source object or in the specified collection (array), but not in both of them.

A version for working with the collection that implements the ICollection<T> interface.

```
void SymmetricExceptWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void SymmetricExceptWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection to produce the symmetrical difference with.

&collection[]

[in] An array to produce the symmetrical difference with.

Note

The result is written to the current collection (array).

UnionWith

Produces the union of the current collection and a passed collection (array). It adds to the current collection (array) missing elements from the specified collection (array).

A version for working with the collection that implements the ICollection<T> interface.

```
void UnionWith(
   ICollection<T>*   collection      // collection
   );
```

A version for working with an array.

```
void UnionWith(
   T&   array[]                      // array
   );
```

Parameters

**collection*

[in] A collection with which the current set will be united.

&collection[]

[in] An array with which the current set will be united.

Note

The result is written to the current collection (array).

IsProperSubsetOf

Determines whether the current sorted set is a proper subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSubsetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSubsetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current sorted set is a proper subset, or false otherwise.

IsProperSupersetOf

Determines whether the current sorted set is a proper superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsProperSupersetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsProperSupersetOf(
   T&   array[]                     // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current sorted set is a proper superset, or false otherwise.

IsSubsetOf

Determines whether the current sorted set is a subset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool IsSubsetOf(
   ICollection<T>*   collection      // a collection to determine the relation
   );
```

A version for working with an array.

```
bool IsSubsetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current sorted set is a subset, or false otherwise.

IsSupersetOf

Determines whether the current sorted set is a superset of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool  IsSupersetOf(
   ICollection<T>*   collection     // a collection to determine the relation
   );
```

A version for working with an array.

```
bool  IsSupersetOf(
   T&   array[]                      // an array to determine the relation
   );
```

Parameters

**collection*

[in] A collection to determine the relation.

&collection[]

[in] An array to determine the relation.

Return Value

Returns true if the current sorted set is a superset, or false otherwise.

Overlaps

Determines whether the current sorted set overlaps the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool Overlaps(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool Overlaps(
   T&   array[]                      // an array to compare
   );
```

Parameters

**collection*

[in] A collection to determine overlapping.

&collection[]

[in] An array to determine overlapping.

Return Value

Returns true if the current sorted set and a collection or an array overlap, or false otherwise.

SetEquals

Determines whether the current sorted set contains all elements of the specified collection or array.

A version for working with the collection that implements the ICollection<T> interface.

```
bool SetEquals(
   ICollection<T>*   collection      // a collection to compare
   );
```

A version for working with an array.

```
bool SetEquals(
   T&   array[]                      // an array to compare
   );
```

Parameters

*collection

[in] A collection to compare elements.

&collection[]

[in] An array to compare elements.

Return Value

Returns true if the current sorted set contains all elements of the specified collection or array, or false otherwise.

GetViewBetween

Gets from the current sorted set a subset specified by the minimum and maximum values.

```
bool GetViewBetween(
   T&   array[],          // an array for writing
   T    lower_value,      // the minimum value
   T    upper_value       // the maximum value
   );
```

Parameters

&array[]

[out] An array for writing the subset.

lower_value

[in] The minimum value of the range.

upper_value

[in] The maximum value of the range.

Return Value

Returns true on successful, or false otherwise.

GetReverse

Gets a copy of the current sorted set, in which all the elements are arranged in a reverse order.

```
bool GetReverse(
   T&   array[]      // an array for writing
   );
```

Parameters

&array[]

[out] An array for writing.

Return Value

Returns true on successful, or false otherwise.

CStack<T>

CStack<T> is a generic class that implements the ICollection<T> interface.

Description

The CStack<T> class is a dynamic collection of T type data, which is organized as a stack that operates on the LIFO (last in, first out) principle.

Declaration

```
template<typename T>
class CStack : public ICollection<T>
```

Header

```
#include <Generic\Stack.mqh>
```

Inheritance Hierarchy

ICollection
    CStack

Class Methods

Method	Description
Add	Adds an element to a stack
Count	Returns the number of elements in a stack
Contains	Determines whether a stack contains an element with the specified value
TrimExcess	Sets the capacity of a stack to the actual number of elements
CopyTo	Copies all elements of a stack to the specified array starting at the specified index
Clear	Removes all elements from a stack
Remove	Removes the first occurrence of the specified element from a stack
Push	Adds an element to a stack
Peek	Returns the head element without removing it from a stack
Pop	Returns the head element and removes it from a stack

Add

Adds an element to the stack.

```
bool Add(
   T  value      // the value of the element
   );
```

Parameters

value

[in] The value of the element to add.

Return Value

Returns true on successful, or false otherwise.

Count

Returns the number of elements in the a stack.

```
int Count();
```

Return Value

Returns the number of elements.

Contains

Determines whether a stack contains an element with the specified value.

```
bool Contains(
   T  item      // the search value
   );
```

Parameters

item

[in] The searched value.

Return Value

Returns true if an element with the specified value is found in the stack, or false otherwise.

TrimExcess

Sets the capacity of a stack to the actual number of elements, and thus frees up unused memory.

```
void TrimExcess();
```

CopyTo

Copies all elements of a stack to the specified array starting at the specified index.

```
int CopyTo(
   T&          dst_array[],      // an array for writing
   const int   dst_start=0       // the starting index for writing
   );
```

Parameters

&dst_array[]

[out] An array to which the elements of the stack will be written.

dst_start=0

[in] An index in the array from which copying starts.

Return Value

Returns the number of copied elements.

Clear

Removes all elements from a stack.

```
void Clear();
```

Remove

Removes the first occurrence of the specified element from a stack.

```
bool Remove(
   T  item      // the element value
   );
```

Parameters

item

[in] The value of the element to be deleted.

Return Value

Returns true on successful, or false otherwise.

Push

Adds an element to the stack.

```
bool Push(
   T  value      // element to add
   );
```

Parameters

value

[in] An element to be added.

Return Value

Returns true on successful, or false otherwise.

Peek

Returns the head element without removing it from a stack.

```
T Peek();
```

Return Value

Returns the head element.

Pop

Returns the head element and removes it from a stack.

```
T Pop();
```

Return Value

Returns the head element.

ArrayBinarySearch

Searches for the specified value in an ascending-sorted one-dimensional array using the IComparable<T> interface to compare elements.

```
template<typename T>
int ArrayBinarySearch(
   T&              array[],          // an array for search
   const int       start_index,      // the starting index
   const int       count,            // the search range
   T               value,            // the search value
   IComparer<T>*   comparer          // interface to compare
   );
```

Parameters

&array[]

[out] The array to search in.

value

[in] The searched value.

**comparer*

[in] An interface for comparing elements.

start_index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

Return Value

Returns the index of the found element. If the search value is not found, it returns the index of the smallest element, which is closest in value.

ArrayIndexOf

Searches for the first occurrence of a value in a one-dimensional array.

```
template<typename T>
int ArrayIndexOf(
   T&          array[],          // an array for search
   T           value,            // the search value
   const int   start_index,      // the starting index
   const int   count             // the search range
   );
```

Parameters

&array[]

[out] The array to search in.

value

[in] The searched value.

start_index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

Return Value

Returns the index of the first found element. If the value is not found, returns -1.

ArrayLastIndexOf

Searches for the last occurrence of a value in a one-dimensional array.

```
template<typename T>
int ArrayLastIndexOf(
   T&          array[],          // an array for search
   T           value,            // the search value
   const int   start_index,      // the starting index
   const int   count             // the search range
   );
```

Parameters

&array[]

[out] The array to search in.

value

[in] The searched value.

start_index

[in] The starting index from which the search begins.

count

[in] The length of the search range.

Return Value

Returns the index of the last found element. If the value is not found, returns -1.

ArrayReverse

Changes the sequence of elements in a one-dimensional array.

```
template<typename T>
bool ArrayReverse(
   T&          array[],          // the source array
   const int   start_index,      // the starting index
   const int   count             // the number of elements
   );
```

Parameters

&array[]

[out] The source array.

start_index

[in] The starting index.

count

[in] The number of array elements participating in the operation.

Return Value

Returns true on successful, or false otherwise.

Compare

Compares the two values, whether one of them is "greater than, less than or equal to" the other one.

A version for comparing two bool values.

```
int Compare(
   const bool   x,          // the first value
   const bool   y           // the second value
   );
```

A version for comparing two char values.

```
int Compare(
   const char   x,          // the first value
   const char   y           // the second value
   );
```

A version for comparing two uchar values.

```
int Compare(
   const uchar   x,         // the first value
   const uchar   y          // the second value
   );
```

A version for comparing two short values.

```
int Compare(
   const short   x,         // the first value
   const short   y          // the second value
   );
```

A version for comparing two ushort values.

```
int Compare(
   const ushort   x,        // the first value
   const ushort   y         // the second value
   );
```

A version for comparing two color values.

```
int Compare(
   const color   x,         // the first value
   const color   y          // the second value
   );
```

A version for comparing two int values.

```
int Compare(
   const int   x,           // the first value
   const int   y            // the second value
   );
```

A version for comparing two uint values.

```
int Compare(
   const uint   x,          // the first value
   const uint   y           // the second value
   );
```

A version for comparing two datetime values.

```
int Compare(
   const datetime   x,      // the first value
   const datetime   y       // the second value
   );
```

A version for comparing two long values.

```
int Compare(
   const long   x,          // the first value
   const long   y           // the second value
   );
```

A version for comparing two ulong values.

```
int Compare(
   const ulong   x,         // the first value
   const ulong   y          // the second value
   );
```

A version for comparing two float values.

```
int Compare(
   const float   x,         // the first value
   const float   y          // the second value
   );
```

A version for comparing two double values.

```
int Compare(
   const double   x,        // the first value
   const double   y         // the second value
   );
```

A version for comparing two string values.

```
int Compare(
   const string   x,        // the first value
   const string   y         // the second value
   );
```

A version for comparing two values of other types.

```
template<typename T>
int Compare(
   T  x,                      // the first value
   T  y                       // the second value
   );
```

Parameters

x

[in] The first value

y

[in] The second value

Return Value

Returns a number that expresses the ratio of the two compared values:

- if the result is less than zero, x is less than y (x<y)
- if the result is equal to zero, x is equal to y (x=y)
- if the result is greater than zero, x is greater than y (x>y)

Note

If the T type is an object that implements the IComparable<T> interface, then the objects will be compared based on its Compare method. In all other cases, 0 is returned.

Equals

Compares two values for equality.

```
template<typename T>
bool Equals(
   T  x,      // the first value
   T  y       // the second value
   );
```

Parameters

x

[in] The first value

y

[in] The second value

Return Value

Returns true if the objects are equal, or false otherwise.

Note

If the T type is an object that implements the IEqualityComparable<T> interface, then the objects will be compared based on its Equals comparison method. The standard comparison for equality is used in all other cases.

GetHashCode

Calculates the hash code value.

A version for working with the bool type.

```
int GetHashCode(
   const bool   value          // value
   );
```

A version for working with the char type.

```
int GetHashCode(
   const char   value          // value
   );
```

A version for working with the uchar type.

```
int GetHashCode(
   const uchar   value         // value
   );
```

A version for working with the short type.

```
int GetHashCode(
   const short   value         // value
   );
```

A version for working with the ushort type.

```
int GetHashCode(
   const ushort   value        // value
   );
```

A version for working with the color type.

```
int GetHashCode(
   const color   value         // value
   );
```

A version for working with the int type.

```
int GetHashCode(
   const int   value           // value
   );
```

A version for working with the uint type.

```
int GetHashCode(
   const uint   value          // value
   );
```

A version for working with the datetime type.

```
int GetHashCode(
   const datetime  value      // value
   );
```

A version for working with the long type.

```
int GetHashCode(
   const long  value         // value
   );
```

A version for working with the ulong type.

```
int GetHashCode(
   const ulong  value        // value
   );
```

A version for working with the float type.

```
int GetHashCode(
   const float  value        // value
   );
```

A version for working with the double type.

```
int GetHashCode(
   const double  value       // value
   );
```

A version for working with the string type.

```
int GetHashCode(
   const string  value       // value
   );
```

A version for working with other types.

```
template<typename T>
int GetHashCode(
   T  value                  // value
   );
```

Parameters

value

[in] The value for which you want to get the hash code.

Return Value

Returns the hash code.

Note

If the T type is an object that implements the IEqualityComparable<T> interface, then the hash code will be obtained based on its HashCode method. In all other cases, the hash code will be calculated as the hash value of the value type name.

File Operations

This section contains the technical details of the file operations classes and descriptions of the corresponding components of the MQL5 standard library.

The file operations classes will save time in developing applications using file input/output operations.

The MQL5 Standard Library (in terms of file operations) is located in the working directory of the terminal in the Include\Files folder.

Class	Description
CFile	Base file operations class
CFileBin	Binary file operations class
CFileTxt	Text file operations class

CFile

CFile is a base class for CFileBin and CFileTxt classes.

Description

CFile class provides the simplified access to MQL5 API file and folder functions for all of its descendants.

Declaration

```
   class CFile: public CObject
```

Title

```
   #include <Files\File.mqh>
```

Inheritance hierarchy

CObject
    CFile

Direct descendants

CFileBin, CFilePipe, CFileTxt

Class Methods by Groups

Attributes	
Handle	Gets file handle
Filename	Gets file name
Flags	Gets file flags
SetUnicode	Sets/clears the FILE_UNICODE flag
SetCommon	Sets/clears the FILE_COMMON flag
General methods for files	
Open	Opens file
Close	Closes file
Delete	Deletes file
IsExist	Checks file for existence
Copy	Copies file
Move	Renames/moves file
Size	Gets file size
Tell	Gets current file position

Attributes	
Seek	Sets current file position
Flush	Flushes data on disk
IsEnding	Checks file for end
IsLineEnding	Checks line for end
General methods for folders	
FolderCreate	Creates folder
FolderDelete	Deletes folder
FolderClean	Clears folder
Files and folders search methods	
FileFindFirst	Begins file search
FileFindNext	Continues file search
FileFindClose	Closes search handle

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Handle

Gets file handle of the opened file.

```
int  Handle()
```

Return Value

Handle of the opened file assigned to the class instance. If there is no file assigned, it returns -1.

FileName

Gets file name of the opened file.

```
string  FileName()
```

Return Value

File name of the opened fil, assigned to the class instance. If there is no file assigned, it returns "".

Flags

Gets flags of the opened file.

```
int  Flags()
```

Return Value

Flags of the opened file assigned to the class instance.

SetUnicode

Sets/clears the FILE_UNICODE flag.

```
void  SetUnicode(
   bool  unicode      // flag value
   )
```

Parameters

unicode

[in] New value for FILE_UNICODE flag.

Note

The result of string operations is dependent on the FILE_UNICODE flag. If it is false, the ANSI codes are used (one byte symbols). If it set, the UNICODE codes are used (two byte symbols). If the file is already opened, the flag cannot be changed.

SetCommon

Sets/clears the FILE_COMMON flag.

```
void  SetCommon(
   bool   common       // flag value
   )
```

Parameters

common

[in] New value for FILE_COMMON flag.

Note

The FILE_COMMON flag determines the current working folder. If it is false, the local terminal folder is used as the current working folder. If it is true, the common data folder is used as the current working folder. If the file is already opened, the flag cannot be changed.

Open

Opens the specified file and, if it successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,        // file name
   int           flags,            // flags
   short         delimiter=9       // separator
   )
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags.

delimiter=9

[in] CSV file separator.

Return Value

Handle of the opened file.

Note

The working folder is dependent on flag that was previously set/reset using the SetCommon() method.

Close

Closes file assigned to the class instance.

```
void  Close()
```

Delete

Deletes the file assigned to the file instance.

```
void  Delete()
```

Delete

Deletes the specified file.

```
void  Delete(
   const string  file_name      // file name
   )
```

Parameters

file_name

[in] File name of the file to delete.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

IsExist

Checks file for existence

```
bool  IsExist(
   const string  file_name      // file name
   )
```

Parameters

file_name

[in] Name of the file to check.

Return Value

true - file exists.

Copy

Copies a file.

```
bool  Copy(
   const string  src_name,       // file name
   int           src_flag,       // flag
   const string  dst_name,       // file name
   int           dst_flags       // flags
   )
```

Parameters

src_name

[in] Name of a source file.

src_flag

[in] Flags of a source file (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Return Value

true - successful, false - cannot copy the file.

Move

Renames/moves file.

```
bool  Move(
   const string  src_name,      // file name
   int           src_flag,      // flag
   const string  dst_name,      // file name
   int           dst_flags      // flags
   )
```

Parameters

src_name

[in] Source file name.

src_flag

[in] Source file flags (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Return Value

true - successful, false - failed to move/rename the file.

Size

Gets file size in bytes.

```
ulong  Size()
```

Return Value

file size in bytes. If there is no file assigned, it returns ULONG_MAX.

Tell

Gets the current file pointer's position.

```
ulong  Tell()
```

Return Value

the current file position. If there no file assigned, it returns ULONG_MAX.

Seek

Sets file pointer's position.

```
void  Seek(
   long                offset,      // offset
   ENUM_FILE_POSITION  origin       // origin
   )
```

Parameters

offset

[in] File offset in bytes (can be negative).

origin

[in] Origin of the offset.

Return Value

true - successful, false - cannot change the file pointer.

Flush

Flushes all of the file input/output buffer data on disk.

```
void  Flush()
```

IsEnding

Checks file for end during the file read operations.

```
bool  IsEnding()
```

Return Value

true - end of file has been achieved during read or seek operation.

IsLineEnding

Checks a text file for end during the file read operations.

```
bool  IsLineEnding()
```

Return Value

true - end of line has been achieved during a txt or csv file read operation (CR-LF chars).

FolderCreate

Creates new folder.

```
bool  FolderCreate(
   const string  folder_name      // folder name
   )
```

Parameters

folder_name

[in] Name of the folder to create. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, false - cannot create the folder.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

FolderDelete

Deletes specified folder.

```
bool  FolderDelete(
   const string  folder_name      // folder name
   )
```

Parameters

folder_name

[in] Name of the folder to delete. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, false - cannot delete the folder.

Note

The working folder is dependent on the flag that was previously set/reset using the SetCommon() method.

FolderClean

Cleans specified folder.

```
bool  FolderClean(
   const string  folder_name      // folder name
   )
```

Parameters

folder_name

[in] Name of the folder to clean. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Return Value

true - successful, and false - cannot change the folder.

Note

The working folder is dependent on the flag previously set/reset by SetCommon() method.

FileFindFirst

Begins file search using the specified filter.

```
int  FileFindFirst(
   const string  filter,         // search filter
   string&       file_name       // reference
   )
```

Parameters

filter

[in] Search filter.

file_name

[out] The reference to the string the name of the first found file is placed into in case of success.

Return Value

The handle that can be used for further file search using FileFindNext, or INVALID_HANDLE if there are no files corresponding to the filter.

Note

The working folder is dependent on the flag previously set/reset by SetCommon() method.

FileFindNext

Continues file search started by the FileFindFirst() method.

```
bool  FileFindNext(
   int       search_handle,      // search handle
   string&   file_name           // reference
   )
```

Parameters

search_handle

[in] Search handle returned by FileFindFirst() method.

file_name

[in] The reference to the string the name of the found file is placed into if successful.

Return Value

true - successful, false - there are no files corresponding to the filter.

FileFindClose

Closes search handle.

```
void  FileFindClose(
   int   search_handle      // search handle
   )
```

Parameters

search_handle

[in] Search handle returned by FileFindFirst() method.

CFileBin

CFileBin is a class for simplified access to binary files.

Description

CFileBin class provides access to binary files.

Declaration

```
class CFileBin: public CFile
```

Title

```
#include <Files\FileBin.mqh>
```

Inheritance hierarchy

CObject
  CFile
    CFileBin

Class Methods by Groups

Open methods	
Open	Opens a binary file
Write methods	
WriteChar	Writes char or uchar type variable
WriteShort	Writes short or ushort type variable
WriteInteger	Writes int or uint type variable
WriteLong	Writes long or ulong type variable
WriteFloat	Writes float type variable
WriteDouble	Writes double type variable
WriteString	Writes string type variable
WriteCharArray	Writes an array of char or uchar type variables
WriteShortArray	Writes an array of short or ushort type variables
WriteIntegerArray	Writes an array of int or uint type variables
WriteLongArray	Writes an array of long or ulong type variables
WriteFloatArray	Writes an array of float variables
WriteDoubleArray	Writes an array of double type variables
WriteObject	Writes data of the CObject class inheritor instance

Open methods	
Read methods	
ReadChar	Reads char or uchar type variable
ReadShort	Reads short or ushort type variable
ReadInteger	Reads int or uint type variable
ReadLong	Reads long or ulong type variable
ReadFloat	Reads float type variable
ReadDouble	Reads double type variable
ReadString	Reads string type variable
ReadCharArray	Reads an array of char or uchar type variables
ReadShortArray	Reads an array of short or ushort type variables
ReadIntegerArray	Reads an array of int or uint type variables
ReadLongArray	Reads an array of long or ulong type variables
ReadFloatArray	Reads an array of float type variables
ReadDoubleArray	Reads an array of double type variables
ReadObject	Reads data of the CObject class inheritor instance

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CFile

Handle, FileName, Flags, SetUnicode, SetCommon, Open, Close, Delete, Size, Tell, Seek, Flush, IsEnding, IsLineEnding, Delete, IsExist, Copy, Move, FolderCreate, FolderDelete, FolderClean, FileFindFirst, FileFindNext, FileFindClose

Open

Opens the specified binary file and, if successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,      // file name
   int           flags           // flags
   )
```

Parameters

file_name

[in] File name of the file to open.

flags

[in] File open flags (the FILE_BIN flag is set forcibly).

Return Value

Handle of the opened file.

WriteChar

Writes char or uchar type variable to file.

```
uint  WriteChar(
   char  value       // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteShort

Writes short or ushort type variable to file.

```
uint  WriteShort(
   short   value       // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteInteger

Writes int or uint type variable to file.

```
uint  WriteInteger(
   int   value      // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteLong

Writes long or ulong type variable to file.

```
uint  WriteLong(
   long  value      // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteFloat

Writes float type variable to file.

```
uint  WriteFloat(
   float  value      // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteDouble

Writes double type variable to file.

```
uint  WriteDouble(
   double  value       // value
   )
```

Parameters

value

[in] Variable to write.

Return Value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value       // value
   )
```

Parameters

value

[in] String to write.

Return Value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value,      // value
   int           size        // size
   )
```

Parameters

value

[in] String to write.

size

[in] Number of bytes to write.

Return Value

Number of bytes written.

WriteCharArray

Writes an array of char or uchar type variables to file.

```
uint  WriteCharArray(
   char&  array[],            // array
   int    start_item=0,       // start element
   int    items_count=-1      // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteShortArray

Writes an array of short or ushort type variables to file.

```
uint  WriteShortArray(
   short&   array[],             // array
   int      start_item=0,        // start element
   int      items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteIntegerArray

Writes an array of int or uint type variables to file.

```
uint  WriteIntegerArray(
   int&   array[],             // array
   int    start_item=0,        // start element
   int    items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteLongArray

Writes an array of long or ulong type variables to file.

```
uint  WriteLongArray(
   long&   array[],             // array
   int     start_item=0,        // start element
   int     items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteFloatArray

Writes an array of float type variables to file.

```
uint  WriteFloatArray(
   float&   array[],             // array
   int      start_item=0,        // start element
   int      items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteDoubleArray

Writes an array of double type variables to file.

```
uint  WriteDoubleArray(
   double&  array[],             // array to write
   int      start_item=0,        // start element
   int      items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - whole array).

Return Value

Number of bytes written.

WriteObject

Writes data of the CObject class inheritor instance to file.

```
bool  WriteObject(
   CObject*   object       // reference to the object
   )
```

Parameters

object

[in] Reference to the CObject class inheritor instance to write.

Return Value

true - successful, false - cannot write the data.

ReadChar

Reads char or uchar type variable from file.

```
bool  ReadChar(
   char&   value       // flag value
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadShort

Reads short or ushort type variable from file.

```
bool  ReadShort(
   short&  value
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadInteger

Reads int or uint type variable from file.

```
bool  ReadInteger(
   int&  value      // variable
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadLong

Reads long or ulong type variable from file.

```
bool  ReadLong(
   long&  value
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadFloat

Reads float type variable from file.

```
bool  ReadFloat(
   float&  value       // variable
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadDouble

Reads double type variable from file.

```
bool  ReadDouble(
   double&  value
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadString

Reads string type variable from file.

```
bool  ReadString(
   string&  value      // string
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadString

Reads string type variable from file.

```
bool  ReadString(
   string&  value
   )
```

Parameters

value

[in] Reference to the variable for placing read data.

Return Value

true - successful, false - cannot read the data.

ReadCharArray

Reads an array of char or uchar type variables from file.

```
bool  ReadCharArray(
   char&  array[],            // array
   int    start_item=0,       // start element
   int    items_count=-1      // number of elements
   )
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadShortArray

Reads an array of short or ushort type variables from file.

```
bool  ReadShortArray(
   short&   array[],            // array
   int      start_item=0,       // start element
   int      items_count=-1      // number of elements
   )
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadIntegerArray

Reads an array of int or uint type variables from file.

```
bool  ReadIntegerArray(
   int&  array[],            // array
   int   start_item=0,       // start element
   int   items_count=-1      // number of elements
   )
```

Parameters

array[]

[in] Reference to the target array of type int or uint.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadLongArray

Reads an array of long or ulong type variables from file.

```
bool  ReadLongArray(
   long&   array[],             // array
   int     start_item=0,        // start element
   int     items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadFloatArray

Reads an array of float type variables from file.

```
bool  ReadFloatArray(
   float&   array[],            // array
   int      start_item=0,       // start element
   int      items_count=-1      // number of elements
   )
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadDoubleArray

Reads an array of double type variables from file.

```
bool  ReadDoubleArray(
   double&  array[],             // array
   int      start_item=0,        // start element
   int      items_count=-1       // number of elements
   )
```

Parameters

array[]

[in] Reference to the variable for placing read data.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Return Value

true - successful, false - cannot read the data.

ReadObject

Reads data of the CObject class inheritor instance from file.

```
bool  ReadObject(
   CObject*   object       // pointer
   )
```

Parameters

object

[in] Pointer to the CObject class inheritor instance to read.

Return Value

true - successful, false - cannot read the data.

CFileTxt

CFileTxt is a class for simplified access to text files.

Description

CFileTxt class provides access to text files.

Declaration

```
class CFileTxt: public CFile
```

Title

```
#include <Files\FileTxt.mqh>
```

Inheritance hierarchy

CObject
  CFile
    CFileTxt

Class Methods by Groups

Open methods	
Open	Opens a text file
Write methods	
WriteString	Writes string type variable
Read methods	
ReadString	Reads string type variable

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CFile

Handle, FileName, Flags, SetUnicode, SetCommon, Open, Close, Delete, Size, Tell, Seek, Flush, IsEnding, IsLineEnding, Delete, IsExist, Copy, Move, FolderCreate, FolderDelete, FolderClean, FileFindFirst, FileFindNext, FileFindClose

Open

Opens the specified text file and, if successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,      // file name
   int           flags           // flags
   )
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags (FILE_TXT flag is forcibly set).

Return Value

Opened file handle.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value      // string
   )
```

Parameters

value

[in] String to write.

Return Value

Number of bytes written.

ReadString

Reads string type variable from file.

```
string  ReadString()
```

Return Value

String which has been read.

String operations

This section contains the technical details of the string operations classes and descriptions of the corresponding components of the MQL5 standard library.

The use of string operations classes will save time in developing applications processing textual data.

The MQL5 standard library (in terms of string operations) is located in the working directory of the terminal in the Include\Strings folder.

Class	Description
CString	Class for string operations

CString

CString is a class for simplified access to the variables of string type.

Description

CString class provides simplified access to MQL5 API functions working with string variables.

Declaration

```
class CString: public CObject
```

Title

```
#include <Strings\String.mqh>
```

Inheritance hierarchy

CObject
    CString

Class Methods by Groups

Data access methods	
Str	Gets a string
Len	Gets length of a string
Copy	Copies a string copy
Fill methods	
Fill	Fills a string
Assign	Assigns a string
Append	Appends a string
Insert	Inserts a string
Compare methods	
Compare	Compares strings
CompareNoCase	Performs a case insensitive string comparison
Substring methods	
Left	Gets a substring from the left
Right	Gets a substring from the right
Mid	Gets a substring from the middle
Trim/delete methods	

Data access methods	
Trim	Trims a string from the left and from the right
TrimLeft	Trims a string from the left
TrimRight	Trims a string from the right
Clear	Clears a string
Convert methods	
ToUpper	Converts a string to uppercase.
ToLower	Converts a string to lowercase.
Reverse	Reverses a string
Search methods	
Find	Searches a substring left to right
FindRev	Searches a substring right to left
Remove	Deletes a substring
Replace	Replaces a substring

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type

Str

Gets a string.

```
string  Str() const;
```

Return Value

Copy of a string.

Len

Gets length of a string.

```
uint  Len() const;
```

Return Value

Length of a string.

Copy

Copies a string by reference.

```
void  Copy(
   string&   copy        // reference
   ) const;
```

Parameters

copy

[in] Reference to a data string.

Copy

Copies a string to the CString class instance.

```
void  Copy(
   CString*   copy       // pointer
   ) const;
```

Parameters

copy

[in] Pointer to CString class instance for data.

Fill

Fills a string with specified char.

```
bool  Fill(
   short  character       // character
   )
```

Parameters

character

[in] Character for filling a string.

Return Value

true - successful, false - cannot fill a string.

Assign

Assigns a string.

```
void  Assign(
   const string  str        // string
   )
```

Parameters

str

[in] String to assign.

Assign

Assigns a string from the CString class instance.

```
void  Assign(
   CString*  str        // pointer
   )
```

Parameters

str

[in] Pointer to the CString class instance to assign.

Append

Appends a string.

```
void  Append(
   const string  str        // string
   )
```

Parameters

str

[in] String to append.

Append

Appends a string from the CString class instance.

```
void  Append(
   CString*  string       // pointer
   )
```

Parameters

string

[in] Pointer to the CString class instance to append.

Insert

Inserts a string to the specified position.

```
uint  Insert(
   uint            pos,      // position
   const string  str       // string
   )
```

Parameters

pos

[in] Insert position.

str

[in] String to insert.

Return Value

Resulted string length.

Insert

Inserts a string to the specified position from the CString class instance.

```
uint  Insert(
   uint       pos,      // position
   CString*   str       // pointer
   )
```

Parameters

pos

[in] Position to insert into.

str

[in] Pointer to the CString class instance to insert.

Return Value

Resulted string length.

Compare

Compares to a string.

```
int  Compare(
   const string  str       // string
   ) const;
```

Parameters

str

[in] String to compare.

Return Value

0 - both strings are equal, -1 - the class string is lower than the string to compare, 1 - the class string is greater than the string to compare.

Compare

Compares to a CString class instance string.

```
int  Compare(
   CString*  str       // pointer
   ) const;
```

Parameters

str

[in] Pointer to CString class instance to compare.

Return Value

0 - strings are equal, -1 - the class string is lower than the string to compare, 1 - the class string is greater than the string to compare.

CompareNoCase

Performs a case insensitive string comparison.

```
int  CompareNoCase(
   const string  str      // string
   ) const;
```

Parameters

str

[in] String to compare.

Return Value

0 - strings are equal, -1 - a class string is lower than a string to compare, 1 - a class string is greater than a string to compare.

CompareNoCase

Compares a string (case insensitive) to a CString class instance string.

```
int  CompareNoCase(
   CString*  str      // pointer
   ) const;
```

Parameters

str

[in] Pointer to CString class instance to compare.

Return Value

0 - if strings are equal, -1 - a class string is lower than a string to compare, 1 - a class string is greater than a string to compare.

Left

Gets a substring of a specified length from the beginning of a string.

```
string  Left(
   uint   count       // length
   )
```

Parameters

count

[in] Substring length.

Return Value

Resulted substring.

Right

Gets a substring of a specified length from the end of a string.

```
string  Right(
   uint   count       // length
   )
```

Parameters

count

[in] Substring length.

Return Value

Resulted substring.

Mid

Gets a substring of a specified length from a specified string position.

```
string  Mid(
   uint   pos,        // position
   uint   count       // length
   )
```

Parameters

pos

[in] Substring position.

count

[in] Substring length.

Return Value

Resulted substring.

Trim

Removes all characters within a set (as well as ' ','\t','\r','\n') at both ends of a string from this string.

```
int  Trim(
   const string  targets      // set
   )
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of removed characters.

Example:

```
//--- example for CString::Trim
#include <Strings\String.mqh>
//---
void OnStart()
  {
   CString str;
   //---
   str.Assign("   \t\tABCD\r\n");
   printf("Source string '%s'",str.Str());
   //---
   str.Trim("DA-DA-DA");
   printf("Result string '%s'",str.Str());
  }
```

TrimLeft

Removes all characters within a set (as well as ' ','\t','\r','\n') at the beginning of a string from this string.

```
int  TrimLeft(
   const string  targets      // set
   )
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of characters removed.

TrimRight

Removes all characters within a set (as well as ' ','\t','\r','\n') at the end of a string from this string.

```
int  TrimRight(
   const string  targets      // set
   )
```

Parameters

targets

[in] Set of characters to remove.

Return Value

Number of characters removed.

Clear

Clears a string.

```
bool  Clear()
```

Return Value

true - successful, false - cannot clear a string.

ToUpper

Converts all string characters to uppercase.

```
bool  ToUpper()
```

Return Value

true - successful, false - cannot convert to uppercase.

ToLower

Converts all string characters to lowercase.

```
bool  ToLower()
```

Return Value

true - successful, false - cannot convert to lowercase.

Reverse

Reverses a string (initial and final characters exchange places pair-wise).

```
void  Reverse()
```

Find

Searches for the first match of a substring from a specified position.

```
int  Find(
   uint           start,         // position
   const string   substring      // substring
   ) const;
```

Parameters

start

[in] Initial position for substring search.

substring

[in] Sample substring to search for.

Return Value

The index of the first match of a substring (-1 - substring is not found).

FindRev

Searches for the last match of a substring.

```
int  FindRev(
   const string  substring      // substring
   ) const;
```

Parameters

substring

[in] Sample substring to search for.

Return Value

The index of the last match of a substring (-1 - substring is not found).

Remove

Removes all substring matches.

```
uint  Remove(
   const string  substring      // substring
   )
```

Parameters

substring

[in] Sample substring to search for.

Return Value

Number of substring removals.

Replace

Replaces all substring matches.

```
uint  Replace(
   const string  substring,      // substring
   const string  newstring       // substring
   )
```

Parameters

substring

[in] Sample substring to search for.

newstring

[in] Sample substring to replace with.

Return Value

Number of substring replacements.

Graphic Objects

This section contains the technical details of working with classes of graphical objects and a description of the relevant components of the MQL5 Standard Library.

The use of classes of graphical objects will save time when creating custom programs (scripts, Expert Advisors).

MQL5 Standard Library (in terms of graphical objects) is located in the working directory of the terminal in the Include\ChartObjects folder.

Class/Group	Description
CChartObject	Base class of a graphic object
Lines	Group classes "Lines"
Channels	Group classes "Channels"
Gann Tools	Group classes "Gann"
Fibonacci Tools	Group classes "Fibonacci"
Elliott Tools	Group classes "Elliott"
Shapes	Group classes "Shapes"
Arrows	Group classes "Arrows"
Controls	Group classes "Controls"

CChartObject

CChartObject is a base class for the classes of chart graphical objects of the Standard MQL5 library.

Description

CChartObject class provides the simplified access to MQL5 API functions for all of its descendants.

Declaration

```
class CChartObject : public CObject
```

Title

```
#include <ChartObjects\ChartObject.mqh>
```

Inheritance hierarchy

CObject

CChartObject

Direct descendants

CChartObjectArrow, CChartObjectBitmap, CChartObjectBmpLabel, CChartObjectCycles, CChartObjectElliottWave3, CChartObjectEllipse, CChartObjectFiboArc, CChartObjectFiboFan, CChartObjectFiboTimes, CChartObjectHLine, CChartObjectRectangle, CChartObjectSubChart, CChartObjectText, CChartObjectTrend, CChartObjectTriangle, CChartObjectVLine

Class Methods by Groups

Attributes	
ChartId	Gets the ID of the chart a graphical object belongs to
Window	Gets the number of a chart window where a graphical object is located
Name	Gets/sets the name of a graphical object
NumPoints	Gets the number of anchor points
Assign	
Attach	Binds a chart graphical object
SetPoint	Sets the anchor point parameters
Delete	
Delete	Deletes a chart graphical object
Detach	Detaches a chart graphical object
Shift	
ShiftObject	The relative object shift

Attributes	
ShiftPoint	The relative object point shift
Object properties	
Time	Gets/sets the time coordinates of an object point
Price	Gets/sets the price coordinate of an object point
Color	Gets/sets the color of the object
Style	Gets/sets the object line style
Width	Gets/sets the object line width
BackGround	Gets/sets the flag of drawing an object in the background
Selected	Gets/sets the "selected" flag of a graphical object
Selectable	Gets/sets the selectable object flag
Description	Gets/sets the text of the object
Tooltip	Gets/sets the tooltip of the object
Timeframes	Gets/sets the mask of the object visibility flags
Z_Order	Gets/sets the graphical object priority for receiving an event of mouse clicking on a chart
CreateTime	Gets the time of the object creation
Object level properties	
LevelsCount	Gets/sets the number of object levels
LevelColor	Gets/sets the level line color
LevelStyle	Gets/sets the level line style
LevelWidth	Gets/sets the level line width
LevelValue	Gets/sets the level
LevelDescription	Gets/sets the level text
Access to API MQL5 functions	
GetInteger	Gets the value of the object property
SetInteger	Sets the value of the object property
GetDouble	Gets the value of the object property
SetDouble	Sets the value of the object property
GetString	Gets the value of the object property
SetString	Sets the value of the object property
Input/Output	

Attributes	
virtual Save	Virtual method of writing to a file
virtual Load	Virtual method of reading from a file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

ChartId

Gets the ID of the chart a graphical object belongs to.

```
long  ChartId() const
```

Return Value

ID of the chart where the graphical object is located. If there is no bound object, it returns -1.

Example:

```
//--- example for CChartObject::ChartId
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get chart idintifier of chart object
   long chart_id=object.ChartId();
  }
```

Window

Gets the index of the chart window where the graphical object is located.

```
int  Window() const
```

Return Value

The number of the chart window where the graphical object is located (0 - main window). If there is no bound object, it returns -1.

Example:

```
//--- example for CChartObject::Window
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get window of chart object
   int window=object.Window();
  }
```

Name (Get Method)

Gets the name of the graphical object.

```
string  Name() const
```

Return Value

Name of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Name (Set Method)

Sets the name of the graphical object.

```
bool  Name(
   string  name      // new name
   )
```

Parameters

name

[in] The new name of the graphical object.

Return Value

true - success, false - cannot change the name.

Example:

```
//--- example for CChartObject::Name
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get name of chart object
   string object_name=object.Name();
   if(object_name!="MyChartObject")
     {
      //--- set name of chart object
      object.Name("MyChartObject");
     }
  }
```

NumPoints

Gets the number of anchor points of a graphical object.

```
int  NumPoints() const
```

Return Value

Number of points linking a graphical object attached to an instance of the class. If there is no attached object, it returns 0.

Example:

```
//--- example for CChartObject::NumPoints
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get points count of chart object
   int points=object.NumPoints();
  }
```

Attach

Attaches a graphical object to an instance of the class.

```
bool  Attach(
   long     chart_id,      // chart ID
   string   name,          // name of the object
   int      window,        // chart window
   int      points         // number of points
   )
```

Parameters

chart_id

[out] Chart identifier.

name

[in] Name of the graphical object.

window

[in] Chart window number (0 - main window).

points

[in] Number of anchor points of the graphical object.

Return Value

true - success, false - cannot bind the object.

Example:

```
//--- example for CChartObject::Attach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- attach chart object
   if(!object.Attach(ChartID(),"MyObject",0,2))
     {
      printf("Object attach error");
      return;
     }
  }
```

SetPoint

Sets new coordinates of the specified anchor point of the graphical object.

```
bool  SetPoint(
   int         point,          // point number
   datetime    new_time,       // time coordinate
   double      new_price       // price coordinate
   )
```

Parameters

point

[in] Number of the graphical object anchor point.

new_time

[in] New value for the time coordinate of the specified anchor point.

new_price

[in] New value for price coordinate of the specified anchor point.

Return Value

true - success, false - cannot change coordinates of the point.

Example:

```
//--- example for CChartObject::SetPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   if(object.NumPoints()>0)
     {
      //--- set point of chart object
      object.SetPoint(0,CurrTime(),price);
     }
  }
```

Delete

Removes an attached graphical object from the chart.

```
bool  Delete()
```

Return Value

true - success, false - cannot remove the object.

Example:

```
//--- example for CChartObject::Delete
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- detach chart object
   if(!object.Delete())
     {
      printf("Object delete error");
      return;
     }
  }
```

Detach

Detaches the graphical object.

```
void  Detach()
```

Return Value

None.

Example:

```
//--- example for CChartObject::Detach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- detach chart object
   object.Detach();
  }
```

ShiftObject

Shifts a graphical object.

```
bool  ShiftObject(
   datetime  d_time,      // increment of time coordinate
   double    d_price      // increment of price coordinate
   )
```

Parameters

d_time

[in] Increment of the time coordinate of all anchor points.

d_price

[in] Increment of the price coordinate of all anchor points.

Return Value

true - success, false - cannot shift the object.

Example:

```
//--- example for CChartObject::ShiftObject
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //--- shift chart object
   object.ShiftObject(d_time,d_price);
  }
```

ShiftPoint

Shifts a specified anchor point of the graphical object.

```
bool  ShiftPoint(
   int        point,        // point number
   datetime   d_time,       // increment of time coordinate
   double     d_price       // increment of price coordinate
   )
```

Parameters

point

[in] Number of a graphical object anchor point.

d_time

[in] Increment of the time coordinate of the specified point.

d_price

[in] Increment of the price coordinate of the specified point.

Return Value

true - success, false - cannot shift the point.

Example:

```
//--- example for CChartObject::ShiftPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //---
   if(object.NumPoints()>0)
     {
      //--- shift point of chart object
      object.ShiftPoint(0,d_time,d_price);
     }
  }
```

Time (Get Method)

Gets the time coordinate of the specified anchor point of a graphical object.

```
datetime  Time(
   int  point       // point number
   ) const
```

Parameters

point

[in] Number of a graphical object anchor point.

Return Value

Time coordinate of the specified anchor point of the graphical object attached to an instance of the class. If there is no attached object or the object does not have this point, it returns 0.

Time (Set Method)

Sets the time coordinate of the specified anchor point of a graphical object.

```
bool  Time(
   int        point,         // point number
   datetime   new_time       // time
   )
```

Parameters

point

[in] Number of a graphical object anchor point.

new_time

[in] New value for the time coordinate of the specified graphical object anchor point.

Return Value

true - success, false - cannot change the time coordinate.

Example:

```
//--- example for CChartObject::Time
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- get time of the chart object point
      datetime point_time=object.Time(i);
      if(point_time==0)
```

```
         {
          //--- set time of the chart object point
          object.Time(i,TimeCurrent());
         }
      }
   }
```

Price (Get Method)

Gets the price coordinate of the specified anchor point of a graphical object.

```
double  Price(
   int  point       // point number
   ) const
```

Parameters

point

[in] Number of a graphical object anchor point.

Return Value

Price coordinate of the specified anchor point of the graphical object attached to an instance of the class. If there is no attached object or the object does not have this point, it returns EMPTY_VALUE.

Price (Set Method)

Sets the price coordinate of the specified anchor point of a graphical object.

```
bool  Price(
   int      point,          // point number
   double   new_price       // price
   )
```

Parameters

point

[in] Number of a graphical object anchor point.

new_price

[in] New value for the price coordinate of the specified graphical object anchor point.

Return Value

true - success, false - cannot change the price coordinate.

Example:

```
//--- example for CChartObject::Price
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- get price of the chart object point
      double point_price=object.Price(i);
```

```
    if(point_price!=price)
      {
       //--- set price for the chart object point
       object.Price(i,price);
      }
   }
}
```

Color (Get Method)

Gets the line color of the graphical object.

```
color  Color() const
```

Return Value

Line color of the graphical object attached to the class instance. If there is no object attached, it returns CLR_NONE.

Color (Set Method)

Sets the line color of the graphical object.

```
bool  Color(
   color  new_color      // new color
   )
```

Parameters

new_color

[in] New value of a graphical object line color.

Return Value

true - success, false - cannot change the color.

Example:

```
//--- example for CChartObject::Color
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get color of chart object
   color object_color=object.Color();
   if(object_color!=clrRed)
     {
     //--- set color of chart object
     object.Color(clrRed);
     }
  }
```

Style (Get Method)

Gets the line style of the graphical object.

```
ENUM_LINE_STYLE  Style() const
```

Return Value

Line style of the graphical object attached to the class instance. If there is no attached object, it returns WRONG_VALUE.

Style (Set Method)

Sets the line style of the graphical object.

```
bool  Style(
   ENUM_LINE_STYLE  new_style      // style
   )
```

Parameters

new_style

[in] New value of the graphical object line style.

Return Value

true - success, false - cannot change the style.

Example:

```
//--- example for CChartObject::Style
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get style of chart object
   ENUM_LINE_STYLE style=object.Style();
   if(style!=STYLE_SOLID)
     {
      //--- set style of chart object
      object.Style(STYLE_SOLID);
     }
  }
```

Width (Get Method)

Gets the line width of the graphical object.

```
int  Width() const
```

Return Value

The line width of the graphical object attached to an instance of the class. If there is no attached object, it returns -1.

Width (Set Method)

Sets the line width of the graphical object.

```
bool  Width(
   int  new_width       // thickness
   )
```

Parameters

new_width

[in] New value of the graphical object line width.

Return Value

true - success, false - cannot change the width.

Example:

```
//--- example for CChartObject::Width
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get width of chart object
   int width=object.Width();
   if(width!=1)
     {
      //--- set width of chart object
      object.Width(1);
     }
  }
```

Background (Get Method)

Gets the flag for drawing a graphical object on the background.

```
bool  Background() const
```

Return Value

Flag for drawing the graphical object, attached to an instance of the class, on the background. If there is no attached object, returns false.

Background (Set Method)

Sets the flag for drawing a graphical object on the background.

```
bool  Background(
   bool  background      // value of the flag
   )
```

Parameters

background

[in] New value of the flag for drawing a graphical object on the background.

Return Value

true - success, false - cannot change the flag.

Example:

```
//--- example for CChartObject::Background
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get background flag of chart object
   bool background_flag=object.Background();
   if(!background_flag)
     {
      //--- set background flag of chart object
      object.Background(true);
     }
  }
```

Selected (Get Method)

Gets the flag indicating that a graphical object is selected. In other words - if the graphical object is selected or not.

```
bool  Selected() const
```

Return Value

The state that the object, attached to an instance of the class, is selected. If there is no attached object, returns false.

Selected (Set Method)

Sets the flag indicating that the graphical object is selected.

```
bool  Selected(
   bool  selected      // value of the flag
   )
```

Parameters

selected

[in] New value of the flag indicating that a graphical object is selected.

Return Value

true - success, false - cannot change the flag.

Example:

```
//--- example for CChartObject::Selected
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get the "selected" flag of chart object
   bool selected_flag=object.Selected();
   if(selected_flag)
     {
     //--- set the "selected" flag of chart object
     object.Selected(false);
     }
  }
```

Selectable (Get Method)

Gets the flag indicating an ability of a graphical object to be selected. In other words - if the graphical object can be selected or not.

```
bool  Selectable() const
```

Return Value

Flag indicating the ability of the graphical object, attached to an instance of the class, to be selected. If there is no attached object, returns false.

Selectable (Set Method)

Sets the flag indicating the ability of a graphical object to be selected.

```
bool  Selectable(
   bool  selectable      // value of the flag
   )
```

Parameters

selectable

[in] New value of the flag indicating an ability of a graphical object to be selected.

Return Value

true - success, false - cannot change the flag.

Example:

```
//--- example for CChartObject::Selectable
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get the "selectable" flag of chart object
   bool selectable_flag=object.Selectable();
   if(selectable_flag)
     {
     //--- set the "selectable" flag of chart object
     object.Selectable(false);
     }
  }
```

Description (Get Method)

Gets a description (text) of a graphical object.

```
string  Description() const
```

Return Value

Description (text) of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Description (Set Method)

Sets the description (text) of the graphical object.

```
bool  Description(
   string  text      // text
   )
```

Parameters

text

[in] New description (text) of a graphical object.

Return Value

true - success, false - cannot change the description (text).

Example:

```
//--- example for CChartObject::Description
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get description of chart object
   string description=object.Description();
   if(description=="")
     {
      //--- set description of chart object
      object.Description("MyObject");
     }
  }
```

Tooltip (Get Method)

Gets the tooltip text of a graphical object.

```
string  Tooltip() const
```

Return Value

The text of a tooltip of the graphical object attached to an instance of the class. If there is no attached object, it returns NULL.

Tooltip (Set Method)

Sets the text of the tooltip of a graphical object.

```
bool  Tooltip(
   string  new_tooltip       // new text of a tooltip
   )
```

Parameters

new_tooltip

[in] New text of a graphical object tooltip.

Return Value

true - success, false - cannot change the tooltip.

Note:

If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value.

Timeframes (Get Method)

Gets visibility flags of a graphical object.

```
int  Timeframes() const
```

Return Value

Visibility flags of the graphical object attached to an instance of the class. If there is no attached object, it returns 0.

Timeframes (Set Method)

Sets visibility flags of a graphical object.

```
bool  Timeframes(
   int  new_timeframes      // visibility flags
   )
```

Parameters

new_timeframes

[in] New visibility flags of the graphical object.

Return Value

true - success, false - cannot change the flags of visibility.

Example:

```
//--- example for CChartObject::Timeframes
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get timeframes of chart object
   int timeframes=object.Timeframes();
   if(!(timeframes&OBJ_PERIOD_H1))
     {
      //--- set timeframes of chart object
      object.Timeframes(timeframes|OBJ_PERIOD_H1);
     }
  }
```

Z_Order (Get Method)

Gets the graphical object priority for receiving an event of mouse clicking on a chart (CHARTEVENT_CLICK).

```
long  Z_Order() const
```

Return Value

Priority of a graphical object, attached to the class instance. If there is no object attached, it returns 0.

Z_Order (Set Method)

Sets the graphical object priority for receiving an event of mouse clicking on a chart (CHARTEVENT_CLICK).

```
bool  Z_Order(
   long  value      // graphical object priority
   )
```

Parameters

value

[in] New value of priority of a graphical object for receiving the event CHARTEVENT_CLICK.

Return Value

true - success, false - cannot change the priority.

Note

Z_Order property manages a priority when handling clicks on graphical objects. By setting the value greater than 0 (default value), you can increase the object priority when handling mouse clicks.

CreateTime

Gets the graphical object creation time.

```
datetime  CreateTime() const
```

Return Value

Creation time of the graphical object attached to the instance of the class. If there is no attached object, it returns 0.

Example:

```
//--- example for CChartObject::CreateTime
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get create time of chart object
   datetime create_time=object.CreateTime();
  }
```

LevelsCount (Get Method)

Gets the number of levels of a graphical object.

```
int  LevelsCount() const
```

Return Value

Number of levels of the graphical object attached to an instance of the class. If there is no attached object, it returns 0.

LevelsCount (Set Method)

Sets the number of levels of the graphical object.

```
bool  LevelsCount(
   int  levels      // number of levels
   )
```

Parameters

levels

[in] The new number of levels of the graphical object.

Return Value

true - success, false - cannot change the number of levels.

Example:

```
//--- example for CChartObject::LevelsCount
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get levels count of chart object
   int levels_count=object.LevelsCount();
   //--- set levels count of chart object
   object.LevelsCount(levels_count+1);
  }
```

LevelColor (Get Method)

Gets the line color of the specified level of a graphical object.

```
color  LevelColor(
   int   level       // level number
   ) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line color of the specified level of the graphical object attached to the instance of the class. If there is no attached object or the object does not have the specified level, it returns CLR_NONE.

LevelColor (Set Method)

Sets the line color of the specified level of the graphical object.

```
bool  LevelColor(
   int     level,          // level number
   color   new_color       // new color
   )
```

Parameters

level

[in] Number of graphical object level.

new_color

[in] New line color of the specified level of a graphical object.

Return Value

true - success, false - cannot change the color.

Example:

```
//--- example for CChartObject::LevelColor
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level color of chart object
      color level_color=object.LevelColor(i);
      if(level_color!=clrRed)
```

```
         {
          //--- set level color of chart object
          object.LevelColor(i,clrRed);
         }
      }
   }
```

LevelStyle (Get Method)

Gets the line style of the specified level of a graphical object.

```
ENUM_LINE_STYLE  LevelStyle(
   int   level       // level number
   ) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line style of the specified level of the graphical object attached to an instance of the class. If there is no attached object or the object does not have the specified level, it returns WRONG_VALUE.

LevelStyle (Set Method)

Sets the line style of the specified level of the graphic object.

```
int  LevelStyle(
   int               level,      // level number
   ENUM_LINE_STYLE   style       // line style
   )
```

Parameters

level

[in] Number of graphical object level.

style

[in] New line style of the specified level of a graphical object.

Return Value

true - success, false - cannot change the style.

Example:

```
//--- example for CChartObject::LevelStyle
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level style of chart object
      ENUM_LINE_STYLE level_style=object.LevelStyle(i);
      if(level_style!=STYLE_SOLID)
```

```
         {
          //--- set level style of chart object
          object.LevelStyle(i,STYLE_SOLID);
         }
      }
   }
```

LevelWidth (Get Method)

Gets the line width of the specified level of a graphical object.

```
int  LevelWidth(
   int  level      // level number
   ) const
```

Parameters

level

[in] Number of graphical object level.

Return Value

Line width of the specified level of the graphical object attached to the instance of the class. If there is no attached object or the object does not have the specified level, it returns -1.

LevelWidth (Set Method)

Sets the line width of the specified level of the graphical object.

```
bool  LevelWidth(
   int  level,          // level number
   int  new_width       // new width
   )
```

Parameters

level

[in] Number of graphical object level.

new_width

[in] New line width of the specified level of a graphical object.

Return Value

true - success, false - cannot change the width.

Example:

```
//--- example for CChartObject::LevelWidth
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level width of chart object
      int level_width=object.LevelWidth(i);
      if(level_width!=1)
```

```
        {
         //--- set level width of chart object
         object.LevelWidth(i,1);
        }
     }
  }
```

LevelValue (Get Method)

Gets the value of the specified level of a graphical object.

```
double  LevelValue(
   int   level      // level number
   ) const
```

Parameters

level

[in] Number of a graphical object level.

Return Value

The value of the specified level of a graphical object that is bound to an instance of the class. If there is no bound object or the object has no level specified, returns EMPTY_VALUE.

LevelValue (Set Method)

Sets the value of the specified level of the graphical object.

```
bool  LevelValue(
   int      level,          // level number
   double   new_value       // new value
   )
```

Parameters

level

[in] Number of a graphical object level.

new_value

[in] New value of the specified level of a graphical object.

Return Value

true - successful, false - cannot change the value.

Example:

```
//--- example for CChartObject::LevelValue
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level value of chart object
      double level_value=object.LevelValue(i);
      if(level_value!=0.1*i)
```

```
         {
          //--- set level value of chart object
          object.LevelValue(i,0.1*i);
         }
      }
   }
```

LevelDescription (Get Method)

Gets a description (text) of the level of a graphical object.

```
string  LevelDescription(
   int   level        // level number
   ) const
```

Parameters

level

[in] Number of a graphical object level.

Return Value

Description (text) of the specified level of a graphical object that is bound to an instance of the class. Returns NULL if there is no bound object or the object has no specified level.

LevelDescription (Set Method)

Sets the description (text) of the specified graphical object level.

```
bool  LevelDescription(
   int      level ,      // level number
   string   text         // text
   )
```

Parameters

level

[in] Number of a graphical object level.

text

[in] New value of description (text) of the specified graphical object level.

Return Value

true - success, false - cannot change the description (text).

Example:

```
//--- example for CChartObject::LevelDescription
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level description of chart object
      string level_description=object.LevelDescription(i);
      if(level_description=="")
```

```
         {
          //--- set level description of chart object
          object.LevelDescription(i,"Level_"+IntegerToString(i));
         }
      }
   }
```

GetInteger

Provides simplified access to the functions of API MQL5 ObjectGetInteger() to receive values of integer properties (of bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color types) of a graphical object bound to a class instance. There are two versions of the function call:

Getting a property value without checking the correctness

```
long  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,          // integer property ID
   int                           modifier=-1       // modifier
   ) const
```

Parameters

prop_id

[in] ID of the graphical object double property.

modifier=-1

[in] Modifier (index) of a double property.

Return Value

Value of an integer property - success, 0 - cannot receive an integer property.

Getting a property value verifying the correctness of the operation

```
bool  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,      // integer property ID
   int                           modifier,     // modifier
   long&                         value         // link to a variable
   ) const
```

Parameters

prop_id

[in] ID of a graphical object integer property.

modifier

[in] Modifier (index) of an integer property.

value

[out] Link to a variable to place an integer property value.

Return Value

true - success, false - cannot get an integer property.

Example:

```
//--- example for CChartObject::GetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
```

```
{
 CChartObject object;
 //--- get color of chart object by easy method
 printf("Objects color is %s",ColorToString(object.GetInteger(OBJPROP_COLOR),true));
 //--- get color of chart object by classic method
 long color_value;
 if(!object.GetInteger(OBJPROP_COLOR,0,color_value))
   {
    printf("Get integer property error %d",GetLastError());
    return;
   }
 else
    printf("Objects color is %s",color_value);
 for(int i=0;i<object.LevelsCount();i++)
   {
    //--- get levels width by easy method
    printf("Level %d width is %d",i,object.GetInteger(OBJPROP_LEVELWIDTH,i));
    //--- get levels width by classic method
    long width_value;
    if(!object.GetInteger(OBJPROP_LEVELWIDTH,i,width_value))
      {
       printf("Get integer property error %d",GetLastError());
       return;
      }
    else
       printf("Level %d width is %d",i,width_value);
   }
}
```

SetInteger

Provides simplified access to the functions of API MQL5 ObjectSetInteger() to change integer properties (of type bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color types) of a graphical object bound to class instance. There are two versions of the function call:

Setting a property value that does not require a modifier

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER    prop_id,      // integer property ID
   long                            value         // value
   )
```

Parameters

prop_id

[in] ID of a graphical object integer property.

value

[in] New value of a changed integer property.

Setting a property value indicating the modifier

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER    prop_id,       // integer property ID
   int                             modifier,      // modifier
   long                            value          // value
   )
```

Parameters

prop_id

[in] ID of a graphical object integer property.

modifier

[in] Modifier (index) of an integer property.

value

[in] New value of an integer property.

Return Value

true - success, false - cannot change the integer property.

Example:

```
//--- example for CChartObject::SetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- set new color of chart object
   if(!object.SetInteger(OBJPROP_COLOR,clrRed))
     {
      printf("Set integer property error %d",GetLastError());
      return;
     }
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- set levels width
      if(!object.SetInteger(OBJPROP_LEVELWIDTH,i,i))
        {
         printf("Set integer property error %d",GetLastError());
         return;
        }
     }
  }
```

GetDouble

Provides simplified access to the functions of API MQL5 ObjectGetDouble() to receive double values (of float and double types) of a graphical object bound to a class instance. There are two versions of the function call:

Getting a property value without checking the correctness

```
double  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,          // integer property ID
   int                          modifier=-1       // modifier
   ) const
```

Parameters

prop_id

[in] ID of the graphical object double property.

modifier=-1

[in] Modifier (index) of a double property.

Return Value

Value of a double property - success, EMPTY_VALUE - cannot receive a double property.

Getting a property value in verifying the correctness of such treatment

```
bool  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,      // double property ID
   int                          modifier,     // modifier
   double&                      value         // link to a variable
   ) const
```

Parameters

prop_id

[in] ID of a graphical object double property.

modifier

[in] Modifier (index) of a double property.

value

[out] Link to a variable to place a double property value.

Return Value

true - success, false - cannot get a double property.

Example:

```
//--- example for CChartObject::GetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
```

```
{
 CChartObject object;
 //---
 for(int i=0;i<object.LevelsCount();i++)
   {
    //--- get levels value by easy method
    printf("Level %d value=%f",i,object.GetDouble(OBJPROP_LEVELVALUE,i));
    //--- get levels value by classic method
    double value;
    if(!object.SetDouble(OBJPROP_LEVELVALUE,i,value))
      {
       printf("Get double property error %d",GetLastError());
       return;
      }
    else
       printf("Level %d value=%f",i,value);
   }
}
```

SetDouble

Provides simplified access to the functions of API MQL5 ObjectSetDouble() to change double properties (of float and double types) of a graphical object bound to a class instance. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,      // double property ID
   double                        value         // value
   )
```

Parameters

prop_id

[in] ID of a graphical object double property.

value

[in] New value of a changed double property.

Setting a property value indicating the modifier

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,       // double property ID
   int                           modifier,      // modifier
   double                        value          // value
   )
```

Parameters

prop_id

[in] ID of a graphical object double property.

modifier

[in] Modifier (index) of a double property.

value

[in] New value of a changed double property.

Return Value

true - success, false - cannot change the double feature.

Example:

```
//--- example for CChartObject::SetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
//---
```

```
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- set level value of chart object
      if(!object.SetDouble(OBJPROP_LEVELVALUE,i,0.1*i))
        {
         printf("Set double property error %d",GetLastError());
         return;
        }
     }
  }
```

GetString

Provides simplified access to the functions of API MQL5 ObjectGetString() for string property values of a graphical object bound to a class instance. There are two versions of a function call:

Getting a property value without checking the correctness

```
string  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,          // string property ID
   int                          modifier=-1       // modifier
   ) const
```

Parameters

prop_id

[in] ID of graphical object string property.

modifier=-1

[in] Modifier (index) of a string property.

Return Value

Value of a string property - success, "" - cannot receive a string property.

Getting a property value verifying the correctness of such treatment

```
bool  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,       // string property ID
   int                          modifier,      // modifier
   string&                      value          // link to a variable
   ) const
```

Parameters

prop_id

[in] ID of a graphical object string property.

modifier

[in] Modifier (index) of a string property.

value

[out] Link to a variable to place a string property value.

Return Value

true - successful, false - cannot get a string property.

Example:

```
//--- example for CChartObject::GetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   string       value;
   //--- get name of chart object by easy method
   printf("Object name is '%s'",object.GetString(OBJPROP_NAME));
   //--- get name of chart object by classic method
   if(!object.GetString(OBJPROP_NAME,0,value))
     {
      printf("Get string property error %d",GetLastError());
      return;
     }
   else
      printf("Object name is '%s'",value);
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get levels description by easy method
      printf("Level %d description is '%s'",i,object.GetString(OBJPROP_LEVELTEXT,i));
      //--- get levels description by classic method
      if(!object.GetString(OBJPROP_LEVELTEXT,i,value))
        {
         printf("Get string property error %d",GetLastError());
         return;
        }
      else
         printf("Level %d description is '%s'",i,value);
     }
  }
```

SetString

Provides simplified access to the functions of API MQL5 ObjectSetString() for changing string properties of a graphical object bound to a class instance. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,      // string property ID
   string                        value         // value
   )
```

Parameters

prop_id

[in] ID of a graphical object string property.

value

[in] New value of a changed string property.

Setting a property value indicating the modifier

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,      // string property ID
   int                           modifier,     // modifier
   string                        value         // value
   )
```

Parameters

prop_id

[in] ID of a graphical object string property.

modifier

[in] Modifier (index) of a string property.

value

[in] New value of a changed string property.

Return Value

true - success, false - cannot change a string property.

Example:

```
//--- example for CChartObject::SetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- set new name of chart object
   if(!object.SetString(OBJPROP_NAME,"MyObject"))
   {
    printf("Set string property error %d",GetLastError());
    return;
   }
 for(int i=0;i<object.LevelsCount();i++)
   {
    //--- set levels description
    if(!object.SetString(OBJPROP_LEVELTEXT,i,"Level_"+IntegerToString(i)))
      {
       printf("Set string property error %d",GetLastError());
       return;
      }
   }
}
```

Save

Saves parameters of the object in the file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] Handle of the file previously opened using the function FileOpen (...).

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CChartObject::Save
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CChartObject object=new CChartObject;
   //--- set object parameters
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

Load

Loads the parameters of the object from the file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the file previously opened using the function FileOpen (...).

Return Value

true - successfully completed, false - error.

Example:

```
//--- example for CChartObject::Load
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CChartObject object;
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use object
   //--- . . .
  }
```

Type

Gets the graphical object type ID.

```
virtual int  Type() const
```

Return Value

Object type ID (0x8888 for CChartObject).

Example:

```
//--- example for CChartObject::Type
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get object type
   int type=object.Type();
  }
```

Line Objects

A group of "Lines" graphical objects.

This section contains the technical details of working with a group of classes of "Lines" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectVLine	"Vertical Line" graphical object
CChartObjectHLine	"Horizontal Line" graphical object
CChartObjectTrend	"Trend Line" graphical object
CChartObjectTrendByAngle	"Trend Line by Angle" graphical object
CChartObjectCycles	"Cyclic Lines" graphical object

See also

Object types, Graphic objects

CChartObjectVLine

CChartObjectVLine is a class for simplified access to "Vertical Line" graphical object properties.

Description

CChartObjectVLine class provides access to "Vertical Line" object properties.

Declaration

```
   class CChartObjectVLine : public CChartObject
```

Title

```
   #include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectVLine

Class Methods by Groups

Create	
Create	Creates "Vertical Line" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Vertical Line" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time           // time coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate of the anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_VLINE for CChartObjectVLine).

CChartObjectHLine

CChartObjectHLine is a class for simplified access to "Horizontal Line" graphical object properties.

Description

CChartObjectHLine class provides access to "Horizonal Line" object properties.

Declaration

```
class CChartObjectHLine : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectHLine

Class Methods by Groups

Create	
Create	Creates "Horizontal Line" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Horizontal Line" graphical object.

```
bool  Create(
   long     chart_id,      // chart identifier
   string   name,          // object name
   long     window,        // chart window
   double   price          // price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_HLINE for CChartObjectHLine).

CChartObjectTrend

CChartObjectTrend is a class for simplified access to "Trend Line" graphical object properties.

Description

CChartObjectTrend class provides access to "Trend Line" object properties.

Declaration

```
   class CChartObjectTrend : public CChartObject
```

Title

```
   #include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectTrend

Direct descendants

CChartObjectChannel, CChartObjectFibo, CChartObjectFiboChannel, CChartObjectFiboExpansion, CChartObjectGannFan, CChartObjectGannGrid, CChartObjectPitchfork, CChartObjectRegression, CChartObjectStdDevChannel, CChartObjectTrendByAngle

Class Methods by Groups

Create	
Create	Creates "Trend Line" graphical object
Properties	
RayLeft	Gets/sets "Ray Left" property
RayRight	Gets/sets "Ray Right" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Graphic objects

Create

Creates "Trend Line" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time1,         // 1st time coordinate
   double      price1,        // 1st price coordinate
   datetime    time2,         // 2nd time coordinate
   double      price2         // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

RayLeft (Get Method)

Gets the value of "Ray Left" property.

```
bool  RayLeft() const
```

Return Value

The value of "Ray Left" property assigned to the class instance. If there is no object assigned, it returns false.

RayLeft (Set Method)

Sets new flag value for the "Ray Left" property.

```
bool  RayLeft(
   bool  ray       // flag
   )
```

Parameters

ray

[in] New value of the "Ray Left" property.

Return Value

true - successful, false - cannot change flag.

RayRight (Get Method)

Gets the value of "Ray Right" property.

```
bool  RayRight() const
```

Return Value

The value of "Ray Right" property, assigned to the class instance. If there is no object assigned, it returns false.

RayRight (Set Method)

Sets new flag value for the "Ray Right" property.

```
bool  RayRight(
   bool  ray      // flag
   )
```

Parameters

ray

[in] New value of the "Ray Right" property.

Return Value

true - successful, false - cannot change flag.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of a file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_TREND for CChartObjectTrend).

CChartObjectTrendByAngle

CChartObjectTrendByAngle is a class for simplified access to "Trend Line by Angle" graphical object properties.

Description

CChartObjectTrendByAngle class provides access to "Trend Line by Angle" object properties.

Declaration

```
class CChartObjectTrendByAngle : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectTrendByAngle

Direct descendants

CChartObjectGannLine

Class Methods by Groups

Create	
Create	Creates "Trend Line by Angle" graphical object
Properties	
Angle	Gets/sets "Angle" property
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Trend Line by Angle" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   long       window,        // chart window
   datetime   time1,         // 1st time coordinate
   double     price1,        // 1st price coordinate
   datetime   time2,         // 2nd time coordinate
   double     price2         // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double  Angle() const
```

Return Value

The value of "Angle" property assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets new value for the "Angle" property.

```
bool  Angle(
   double  angle      // angle
   )
```

Parameters

angle

[in] New value of the "Angle" property.

Return Value

true - successful, false - cannot change the property.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_TRENDBYANGLE for CChartObjectTrendByAngle).

CChartObjectCycles

CChartObjectCycles is a class for simplified access to "Cyclic Lines" graphical object properties.

Description

CChartObjectCycles provides access to "Cyclic Lines" object properties.

Declaration

```
class CChartObjectCycles : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectCycles

Class Methods by Groups

Create	
Create	Creates "Cycle Lines" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Cyclic Lines" graphical object.

```
bool  Create(
   long       chart_id,       // chart identifier
   string     name,           // object name
   long       window,         // chart window
   datetime   time1,          // 1st time coordinate
   double     price1,         // 1st price coordinate
   datetime   time2,          // 2nd time coordinate
   double     price2          // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_CYCLES for CChartObjectCycles).

Channel Objects

A group of "Channels" graphical objects.

This section contains the technical details of working with a group of classes of "Channels" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectChannel	"Equidistant Channel" graphical object
CChartObjectRegression	"Linear Regression Channel" graphical object
CChartObjectStdDevChannel	"Standard deviations Channel" graphical object
CChartObjectPitchfork	"Andrew's Pitchfork" graphical object

See also

Object types, Graphic objects

CChartObjectChannel

CChartObjectChannel is a class for simplified access to "Equidistant Channel" graphical object properties.

Description

CChartObjectChannel class provides access to "Equidistant Channel" object properties.

Declaration

```
   class CChartObjectChannel : public CChartObjectTrend
```

Title

```
   #include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectChannel

Class Methods by Groups

Create	
Create	Creates "Equidistant Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Equidistant Channel" graphic object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time1,         // time coordinate of the first anchor point
   double      price1,        // price coordinate of the first anchor point
   datetime    time2,         // time coordinate of the second anchor point
   double      price2,        // price coordinate of the second anchor point
   datetime    time3,         // time coordinate of the third anchor point
   double      price3         // price coordinate of the third anchor point
   )
```

Parameters

chart_id

[in] Chart ID (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_CHANNEL for CChartObjectChannel).

CChartObjectRegression

CChartObjectRegression is a class for simplified access to "Linear Regression Channel" graphical object properties.

Description

CChartObjectRegression class provides access to "Linear Regression Channel" object properties.

Declaration

```
class CChartObjectRegression : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectRegression

Class Methods by Groups

Create	
Create	Creates "Linear Regression Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Linear Regression Channel" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   long        window,        // chart window
   datetime    time1,         // first time coordinate
   datetime    time2          // second time coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_REGRESSION for CChartObjectRegression).

CChartObjectStdDevChannel

CChartObjectStdDevChannel is a class for simplified access to "Standard Deviation Channel" graphical object properties.

Description

CChartObjectStdDevChannel class provides access to "Standard Deviation Channel" object properties.

Declaration

```
class CChartObjectStdDevChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectStdDevChannel

Class Methods by Groups

Create	
Create	Creates "Standard Deviation Channel" graphical object
Properties	
Deviations	Gets/sets "Deviation" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

See also

Object types, Graphic objects

Create

Creates "Standard Deviation Channel" graphical object.

```
bool  Create(
   long       chart_id,       // chart identifier
   string     name,           // object name
   int        window,         // chart window
   datetime   time1,          // first time coordinate
   datetime   time2,          // second time coordinate
   double     deviation       // deviation
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

deviation

[in] Numerical value for "Deviation" property.

Return Value

true - successful, false - error.

Deviation (Get Method)

Gets "Deviation" property value.

```
double  Deviation() const
```

Return Value

Value of "Deviation" property assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Deviation (Set Method)

Sets "Deviation" property value.

```
bool  Deviation(
   double  deviation      // deviation
   )
```

Parameters

deviation

[in] New value for "Deviation" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_STDDEVCHANNEL for CChartObjectStdDevChannel).

CChartObjectPitchfork

CChartObjectPitchfork is a class for simplified access to "Andrew's Pitchfork" graphical object properties.

Description

CChartObjectPitchfork class provides access to "Andrew's Pitchfork" object properties.

Declaration

```
class CChartObjectPitchfork : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectTrend
      CChartObjectPitchfork

Class Methods by Groups

Create	
Create	Creates "Andrew's Pitchfork" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Andrew's Pitchfork" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   long        window,        // chart window
   datetime    time1,         // time coordinate of the first anchor point
   double      price1,        // price coordinate of the first anchor point
   datetime    time2,         // time coordinate of the second anchor point
   double      price2,        // price coordinate of the second anchor point
   datetime    time3,         // time coordinate of the third anchor point
   double      price3         // price coordinate of the third anchor point
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] Unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_PITCHFORK for CChartObjectPitchfork).

Gann Tools

A group "Gann Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Gann Tools" graphical objects and the description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectGannLine	"Gann Line" graphical objects
CChartObjectGannFan	"Gann Fan" graphical objects
CChartObjectGannGrid	"Gann Grid" graphical objects

See also

Object types, Graphic objects

CChartObjectGannLine

CChartObjectGannLine is a class for simplified access to "Gann Line" graphical object properties.

Description

CChartObjectGannLine class provides access to "Gann Line" object properties.

Declaration

```
class CChartObjectGannLine : public CChartObjectTrendByAngle
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectTrendByAngle
        - CChartObjectGannLine

Class Methods by Groups

Create	
Create	Creates "Gann Line" graphical objects
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

Methods inherited from class CChartObjectTrendByAngle

Angle, Angle, Create

See also

Object types, Graphic objects

Create

Creates "Gann Line" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time1,         // first time coordinate
   double      price1,        // first price coordinate
   datetime    time2,         // second time coordinate
   double      ppb            // pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb      // pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_GANNLINE for CChartObjectGannLine).

CChartObjectGannFan

CChartObjectGannFan is a class for simplified access to "Gann Fan" graphical object properties.

Description

CChartObjectGannFan class provides access to "Gann Fan" object properties.

Declaration

```
class CChartObjectGannFan : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy

CObject
    CChartObject
        CChartObjectTrend
            CChartObjectGannFan

Class Methods by Groups

Create	
Create	Creates "Gann Fan" graphical object
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Downtrend	Gets/sets "Downtrend" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

See also

Object types, Graphic objects

Create

Creates "Gann Fan" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     ppb            // pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb        // pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Downtrend (Get Method)

Gets the value of "Downtrend" flag.

```
bool  Downtrend() const
```

Return Value

Value of the "Downtrend" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool  Downtrend(
   bool  downtrend      // flag value
   )
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Return Value

true - successful, false - cannot change the flag.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_GANNFAN for CChartObjectGannFan).

CChartObjectGannGrid

CChartObjectGannGrid is a class for simplified access to "Gann Grid" graphical object properties.

Description

CChartObjectGannGrid class provides access to "Gann Grid" object properties.

Declaration

```
class CChartObjectGannGrid : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectGannGrid

Class Methods by Groups

Create	
Create	Creates "Gann Grid" graphical object
Properties	
PipsPerBar	Gets/sets "Pips per bar" property
Downtrend	Gets/sets "Downtrend" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create

See also

Object types, Graphic objects

Create

Creates "Gann Grid" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     ppb            // pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Return Value

true - successful, false - error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Return Value

Value of "Pips per bar" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb      // Pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Return Value

true - successful, false - cannot change the property.

Downtrend (Get Method)

Gets the value of "Downtrend" property.

```
bool  Downtrend() const
```

Return Value

Value of "Downtrend" property of the object assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool  Downtrend(
   bool  downtrend      // flag value
   )
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_GANNGRID for CChartObjectGannGrid).

Fibonacci Tools

A group of "Fibonacci Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Fibonacci Tools" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectFibo	"Fibonacci Retracement" graphical object
CChartObjectFiboTimes	"Fibonacci Time Zones" graphical object
CChartObjectFiboFan	"Fibonacci Fan" graphical object
CChartObjectFiboArc	"Fibonacci Arc" graphical object
CChartObjectFiboChannel	"Fibonacci Channel" graphical object
CChartObjectFiboExpansion	"Fibonacci Expansion" graphical object

See also

Object types, Graphic objects

CChartObjectFibo

CChartObjectFibo is a class for simplified access to "Fibonacci Retracement" graphical object properties.

Description

CChartObjectFibo class provides access to "Fibonacci Retracement" object properties.

Declaration

```
   class CChartObjectFibo : public CChartObjectTrend
```

Title

```
   #include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectFibo

Class Methods by Groups

Create	
Create	Creates "Fibonacci Retracement" graphical objects
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Fibonacci Retracement" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2         // second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_FIBO for CChartObjectFibo).

CChartObjectFiboTimes

CChartObjectFiboTimes is a class for simplified access to "Fibonacci Time Zones" graphical object properties.

Description

CChartObjectFiboTimes class provides access to "Fibonacci Time Zones" object properties.

Declaration

```
class CChartObjectFiboTimes : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectFiboTimes

Class Methods by Groups

Create	
Create	Creates "Fibonacci Time Zones" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Fibonacci Time Zones" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2         // second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_FIBOTIMES for CChartObjectFiboTimes).

CChartObjectFiboFan

CChartObjectFiboFan is a class for simplified access to "Fibonacci Fan" graphical object properties.

Description

CChartObjectFiboFan class provides access to "Fibonacci Fan" object properties.

Declaration

```
class CChartObjectFiboFan : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectFiboFan

Class Methods by Groups

Create	
Create	Creates "Fibonacci Fan" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Fibonacci Fan" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2         // second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_FIBOFAN for CChartObjectFiboFan).

CChartObjectFiboArc

CChartObjectFiboArc is a class for simplified access to "Fibonacci Arc" graphical object properties.

Description

CChartObjectFiboArc class provides access to "Fibonacci Arc" object properties.

Declaration

```
class CChartObjectFiboArc : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectFiboArc

Class Methods by Groups

Create	
Create	Creates "Fibonacci Arc" graphical object
Properties	
Scale	Gets/sets "Scale" property
Ellipse	Gets/sets "Ellipse" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Graphic objects

Create

Creates "Fibonacci Arc" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2,        // second price coordinate
   double     scale          // scale
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

scale

[in] Scale.

Return Value

true - successful, false - error.

Scale (Get Method)

Gets the value of "Scale" property.

```
double  Scale() const
```

Return Value

Value of "Scale" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   double  scale      // scale
   )
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

Ellipse (Get Method)

Gets the value of "Ellipse" flag.

```
bool  Ellipse() const
```

Return Value

Value of "Ellipse" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

Ellipse (Set Method)

Sets "Ellipse" flag value.

```
bool  Ellipse(
   bool  ellipse      // flag value
   )
```

Parameters

ellipse

[in] New value for "Ellipse" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_FIBOARC for CChartObjectFiboArc).

CChartObjectFiboChannel

CChartObjectFiboChannel is a class for simplified access to "Fibonacci Channel" graphical object properties.

Description

CChartObjectFiboChannel class provides access to "Fibonacci Channel" object properties.

Declaration

```
class CChartObjectFiboChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectFiboChannel

Class Methods by Groups

Create	
Create	Creates "Fibonacci Channel" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Fibonacci Channel" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2,        // second price coordinate
   datetime   time3,         // third time coordinate
   double     price3         // third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_FIBOCHANNEL for CChartObjectFiboChannel).

CChartObjectFiboExpansion

CChartObjectFiboExpansion is a class for simplified access to "Fibonacci Expansion" graphical object properties.

Description

CChartObjectFiboExpansion class provides access to "Fibonacci Expansion" object properties.

Declaration

```
   class CChartObjectFiboExpansion : public CChartObjectTrend
```

Title

```
   #include <ChartObjects\ChartObjectsFibo.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTrend
      - CChartObjectFiboExpansion

Class Methods by Groups

Create	
Create	Creates "Fibonacci Expansion" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectTrend

RayLeft, RayLeft, RayRight, RayRight, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Fibonacci Expansion" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time1,         // first time coordinate
   double      price1,        // first price coordinate
   datetime    time2,         // second time coordinate
   double      price2,        // second price coordinate
   datetime    time3,         // third time coordinate
   double      price3         // third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_EXPANSION for CChartObjectFiboExpansion).

Elliott Tools

A group of "Elliott Tools" graphical objects.

This section contains the technical details of working with a group of classes of "Elliott Tools" graphical objects.

Class name	Object
CChartObjectElliottWave3	"Correcting Wave" graphical object
CChartObjectElliottWave5	"Impulse Wave" graphical object

See also

Object types, Graphic objects

CChartObjectElliottWave3

CChartObjectElliottWave3 is a class for simplified access to "Correcting Wave" graphical object properties.

Description

CChartObjectElliottWave3 class provides access to "Correcting Wave" object properties.

Declaration

```
class CChartObjectElliottWave3 : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectElliottWave3

Direct descendants

CChartObjectElliottWave5

Class Methods by Groups

Create	
Create	Creates "Correcting Wave" graphical object
Properties	
Degree	Gets/sets "Degree" property
Lines	Gets/sets "Lines" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Graphic objects

Create

Creates "Correcting Wave" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time1,         // first time coordinate
   double      price1,        // first price coordinate
   datetime    time2,         // second time coordinate
   double      price2,        // second price coordinate
   datetime    time3,         // third time coordinate
   double      price3         // third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Time coordinate of the third anchor point.

Return Value

true - successful, false - error.

Degree (Get Method)

Gets the value of "Degree" property.

```
ENUM_ELLIOT_WAVE_DEGREE  Degree() const
```

Return Value

Value of "Degree" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Degree (Set Method)

Sets new value for "Degree" property.

```
bool  Degree(
   ENUM_ELLIOT_WAVE_DEGREE  degree      // property value
   )
```

Parameters

degree

[in] New value for "Degree" property.

Return Value

true - successful, false - cannot change the property.

Lines (Get Method)

Gets the value of "Lines" property.

```
bool  Lines() const
```

Return Value

Value of "Lines" property of the object assigned to the class instance. If there is no object assigned, it returns false.

Lines (Set Method)

Sets new value for "Lines" property.

```
bool  Lines(
   bool  lines      // flag value
   )
```

Parameters

lines

[in] New value for "Lines" property.

Return Value

true - successful, false - cannot change the flag.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using the FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_ELLIOTWAVE3 for CChartObjectElliottWave3).

CChartObjectElliottWave5

CChartObjectElliottWave5 is a class for simplified access to "Impulse Wave" graphical object properties.

Description

CChartObjectElliottWave5 class provides access to "Impulse Wave" object properties.

Declaration

```
   class CChartObjectElliottWave5 : public CChartObjectElliottWave3
```

Title

```
   #include <ChartObjects\ChartObjectsElliott.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectElliottWave3
      - CChartObjectElliottWave5

Class Methods by Groups

Create	
Create	Creates "Impulse Wave" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectElliottWave3

Degree, Degree, Lines, Lines, Create, Save, Load

See also

Object types, Graphic objects

Create

Creates "Impulse Wave" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2,        // second price coordinate
   datetime   time3,         // third time coordinate
   double     price3,        // third price coordinate
   datetime   time4,         // fourth time coordinate
   double     price4,        // fourth price coordinate
   datetime   time5          // fifth time coordinate
   double     price5,        // fifth price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

time4

[in] Time coordinate of the fourth anchor point.

price4

[in] Price coordinate of the fourth anchor point.

time5

[in] Time coordinate of the fifth anchor point.

price5

[in] Price coordinate of the fifth anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_ELLIOTWAVE5 for CChartObjectElliottWave5).

Shape Objects

A group of "Shapes" graphical objects.

This section contains the technical details of working with a group of classes of "Shapes" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectRectangle	"Rectangle" graphical objects
CChartObjectTriangle	"Triangle" graphical objects
CChartObjectEllipse	"Ellipse" graphical objects

See also

Object types, Graphic objects

CChartObjectRectangle

CChartObjectRectangle is a class for simplified access to "Rectangle" graphical object properties.

Description

CChartObjectRectangle class provides access to "Rectangle" object properties.

Declaration

```
class CChartObjectRectangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectRectangle

Class Methods by Groups

Create	
Create	Creates "Rectangle" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Rectangle" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   long       window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2         // second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_RECTANGLE for CChartObjectRectangle).

CChartObjectTriangle

CChartObjectTriangle is a class for simplified access to "Triangle" graphical object properties.

Description

CChartObjectTriangle class provides access to "Triangle" object properties.

Declaration

```
class CChartObjectTriangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectTriangle

Class Methods by Groups

Create	
Create	Creates "Triangle" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Triangle" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   long       window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2,        // second price coordinate
   datetime   time3,         // third time coordinate
   double     price3         // third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_TRIANGLE for CChartObjectTriangle).

CChartObjectEllipse

CChartObjectEllipse is a class for simplified access to "Ellipse" graphical object properties.

Description

CChartObjectEllipse class provides access to "Ellipse" object properties.

Declaration

```
class CChartObjectEllipse : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectEllipse

Class Methods by Groups

Create	
Create	Creates "Ellipse" graphical object
Input/output	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint, Save, Load

See also

Object types, Graphic objects

Create

Creates "Ellipse" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time1,         // first time coordinate
   double     price1,        // first price coordinate
   datetime   time2,         // second time coordinate
   double     price2,        // second price coordinate
   datetime   time3,         // third time coordinate
   double     price3         // third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
int  Type() const
```

Return Value

Object type identifier (OBJ_ELLIPSE for CChartObjectEllipse).

Arrow Objects

Group for Arrows graphical objects.

This section contains the technical details of working with a group of classes of "Arrow" graphical objects and a description of the relevant components of the MQL5 Standard Library. In essence, the arrow is a certain icon which is associated with a specific code. There are two types of "Arrow" graphical object to display icons on the charts:

- "Arrow" object, which allows you to specify the code of the icon displayed by the object.
- Group of objects to display a certain type of icons (corresponding to a certain fixed code).

Class for working with arrows displaying arbitrary code icons

Class name	Name of the arrow object
CChartObjectArrow	Arrow

Classes for working arrows displaying a fixed code icon

Class name	Name of the arrow object
CChartObjectArrowCheck	Check
CChartObjectArrowDown	Arrow Up
CChartObjectArrowUp	Arrow Down
CChartObjectArrowStop	Stop Sign
CChartObjectArrowThumbDown	Thumbs Up
CChartObjectArrowThumbUp	Thumbs Down
CChartObjectArrowLeftPrice	Left Price Label
CChartObjectArrowRightPrice	Right Price Label

See also

Object types, Methods of Object Binding, Graphic objects

CChartObjectArrow

CChartObjectArrow is a class for simplified access to "Arrow" graphical object properties.

Description

CChartObjectArrow class provides access to common properties of "Arrow" objects to all of its descendants.

Declaration

```
class CChartObjectArrow : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

Inheritance hierarchy

CObject

CChartObject

CChartObjectArrow

Direct descendants

CChartObjectArrowCheck, CChartObjectArrowDown, CChartObjectArrowLeftPrice, CChartObjectArrowRightPrice, CChartObjectArrowStop, CChartObjectArrowThumbDown, CChartObjectArrowThumbUp, CChartObjectArrowUp

Class Methods by Groups

Create	
Create	Creates "Arrow" graphical object
Properties	
ArrowCode	Gets/sets "Arrow Code" property
Anchor	Gets/sets "Anchor" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Methods of objects binding, Graphic objects

Create

Creates "Arrow" graphical object.

```
bool  Create(
   long        chart_id,       // chart ID
   string      name,           // object Name
   int         window,         // chart Window
   datetime    time,           // time
   double      price,          // price
   char        code            // arrow code
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique object name.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate.

price

[in] Price coordinate.

code

[in] "Arrow" code (Wingdings).

Return Value

true - success, false - error.

Example:

```
//--- example for CChartObjectArrow::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
```

```
        return;
       }
//--- use arrow
//--- . . .
  }
```

ArrowCode (Get Method)

Gets "Arrow" symbol code.

```
char  ArrowCode() const
```

Return Value

"Arrow" symbol code of the object assigned to the class instance. If there is no object assigned, it returns 0.

ArrowCode (Set Method)

Sets symbol code for "Arrow"

```
bool  ArrowCode(
   char  code      // code value
   )
```

Parameters

code

[in] New value for "arrow" code (Wingdings).

Return Value

true - success, false - cannot change the code.

Example:

```
//--- example for CChartObjectArrow::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   char              code=181;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,code))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- change the code of arrow
//--- . . .
//--- get code of arrow
   if(arrow.ArrowCode()!=code)
     {
     //--- set code of arrow
```

```
      arrow.ArrowCode(code);
      }
//--- use arrow
//--- . . .
   }
```

Anchor (Get Method)

Gets anchor type of the "Arrow" object

```
ENUM_ARROW_ANCHOR  Anchor() const
```

Return Value

Anchor type of the "Arrow" object assigned to the class instance (to the chart). If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets anchor type for the "Arrow" object

```
bool  Anchor(
   ENUM_ARROW_ANCHOR  anchor      // anchor type
   )
```

Parameters

anchor

[in] New anchor type value

Return Value

true - successful, false - cannot change the anchor type.

Example:

```
//--- example for CChartObject::Anchor
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- get anchor of arrow
   if(arrow.Anchor()!=anchor)
     {
     //--- set anchor of arrow
     arrow.Anchor(anchor);
     }
//--- use arrow
//--- . . .
  }
```

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of file opened previously using the FileOpen(...) function.

Return Value

true - successful, false - error.

Example:

```
//--- example for CChartObjectArrow::Save
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int                file_handle;
   CChartObjectArrow arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of file previously opened using the FileOpen(...) function.

Return Value

true - success, false - error.

Example:

```
//--- example for CChartObjectArrow::Load
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int               file_handle;
   CChartObjectArrow arrow;
//--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
//--- use arrow
//--- . . .
  }
```

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (for example, OBJ_ARROW for CChartObjectArrow)

Example:

```
//--- example for CChartObjectArrow::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   //--- get arrow type
   int type=arrow.Type();
  }
```

Arrows with fixed code

"Arrows with fixed code" are classes for simplified access to the properties of the following graphical objects:

Class name	Arrow object name
CChartObjectArrowCheck	"Arrow Check"
CChartObjectArrowDown	"Arrow Down"
CChartObjectArrowUp	"Arrow Up"
CChartObjectArrowStop	"Arrow Stop"
CChartObjectArrowThumbDown	"Good" ("Thumbs up")
CChartObjectArrowThumbUp	"Bad" ("Thumbs down")
CChartObjectArrowLeftPrice	"Left price" arrow
CChartObjectArrowRightPrice	"Right price" arrow

Description

"Arrows with fixed code" classes provide access to the object properties.

Declarations

```
class CChartObjectArrowCheck      : public CChartObjectArrow;
class CChartObjectArrowDown       : public CChartObjectArrow;
class CChartObjectArrowUp         : public CChartObjectArrow;
class CChartObjectArrowStop       : public CChartObjectArrow;
class CChartObjectArrowThumbDown  : public CChartObjectArrow;
class CChartObjectArrowThumbUp    : public CChartObjectArrow;
class CChartObjectArrowLeftPrice  : public CChartObjectArrow;
class CChartObjectArrowRightPrice : public CChartObjectArrow;
```

Title

```
<ChartObjects\ChartObjectsArrows.mqh>
```

Class Methods by Groups

Create	
Create	Creates the graphical object matching the class
Properties	
ArrowCode	"Stub" for symbol code change method
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Methods of object binding, Graphic objects

Create

Creates "Arrow with fixed code" graphical object.

```
bool  Create(
   long       chart_id,      // chart ID
   string     name,          // object Name
   int        window,        // chart Window
   datetime   time,          // time
   double     price          // price
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] Unique name of the object to create.

window

[in] Chart window number (0 - main window).

time

[in] Time coordinate.

price

[in] Price coordinate.

Return Value

true - successful, false - error.

Example:

```
//--- example for CChartObjectArrowCheck::Create
//--- example for CChartObjectArrowDown::Create
//--- example for CChartObjectArrowUp::Create
//--- example for CChartObjectArrowStop::Create
//--- example for CChartObjectArrowThumbDown::Create
//--- example for CChartObjectArrowThumbUp::Create
//--- example for CChartObjectArrowLeftPrice::Create
//--- example for CChartObjectArrowRightPrice::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
```

```
      {
       //--- arrow create error
       printf("Arrow create: Error %d!",GetLastError());
       //---
       return;
      }
//--- use arrow
//--- . . .
  }
```

ArrowCode

Prohibits "Arrow" symbol code changes.

```
bool  ArrowCode(
   char  code       // code value
   )
```

Parameters

code

[in] Any value

Return Value

Always false.

Example:

```
//--- example for CChartObjectArrowCheck::ArrowCode
//--- example for CChartObjectArrowDown::ArrowCode
//--- example for CChartObjectArrowUp::ArrowCode
//--- example for CChartObjectArrowStop::ArrowCode
//--- example for CChartObjectArrowThumbDown::ArrowCode
//--- example for CChartObjectArrowThumbUp::ArrowCode
//--- example for CChartObjectArrowLeftPrice::ArrowCode
//--- example for CChartObjectArrowRightPrice::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- set code of arrow
   if(!arrow.ArrowCode(181))
     {
      //--- it is not error
      printf("Arrow code can not be changed");
     }
//--- use arrow
//--- . . .
  }
```

Type

Returns graphical object type identifier

```
virtual int  Type() const
```

Return Value

Object type identifier:

CChartObjectArrowCheck - OBJ_ARROW_CHECK,

CChartObjectArrowDown - OBJ_ARROW_DOWN,

CChartObjectArrowUp - OBJ_ARROW_UP,

CChartObjectArrowStop - OBJ_ARROW_STOP,

CChartObjectArrowThumbDown - OBJ_ARROW_THUMB_DOWN,

CChartObjectArrowThumbUp - OBJ_ARROW_THUMB_UP,

CChartObjectArrowLeftPrice - OBJ_ARROW_LEFT_PRICE,

CChartObjectArrowRightPrice - OBJ_ARROW_RIGHT_PRICE.

Example:

```
//--- example for CChartObjectArrowCheck::Type
//--- example for CChartObjectArrowDown::Type
//--- example for CChartObjectArrowUp::Type
//--- example for CChartObjectArrowStop::Type
//--- example for CChartObjectArrowThumbDown::Type
//--- example for CChartObjectArrowThumbUp::Type
//--- example for CChartObjectArrowLeftPrice::Type
//--- example for CChartObjectArrowRightPrice::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- get arrow type
   int type=arrow.Type();
  }
```

Control Objects

A group of "Object Controls" graphical objects.

This section contains the technical details of working with a group of classes of "Object Controls" graphical objects and a description of the relevant components of the MQL5 Standard Library.

Class name	Object
CChartObjectText	"Text" graphical object
CChartObjectLabel	"Text Label" graphical object
CChartObjectEdit	"Edit" graphical object
CChartObjectButton	"Button" graphical object
CChartObjectSubChart	"Chart" graphical object
CChartObjectBitmap	"Bitmap" graphical object
CChartObjectBmpLabel	"Bitmap Label" graphical object
CChartObjectRectLabel	"Rectangle Label" graphical object

See also

Object types, Graphic objects

CChartObjectText

CChartObjectText is a class for simplified access to "Text" graphical object properties.

Description

CChartObjectText class provides access to "Text" object properties.

Declaration

```
class CChartObjectText : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectText

Direct descendants

CChartObjectLabel

Class Methods by Groups

Create	
Create	Creates "Text" graphical object
Properties	
Angle	Gets/sets "Angle" property
Font	Gets/sets "Font" property
FontSize	Gets/sets "FontSize" property
Anchor	Gets/sets "Anchor" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Derived classes:

- CChartObjectLabel

See also

Object types, Object properties, Methods of object binding, Graphic objects

Create

Creates "Text" graphical object.

```
bool  Create(
   long        chart_id,      // chart identifier
   string      name,          // object name
   int         window,        // chart window
   datetime    time,          // time coordinate
   double      price          // price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double  Angle() const
```

Return Value

Value of "Angle" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets a value for "Angle" property.

```
bool  Angle(
   double  angle      // property value
   )
```

Parameters

angle

[in] New value for "Angle" property.

Return Value

true - successful, false - cannot change the property.

Font (Get Method)

Gets the value of "Font" property.

```
string  Font() const
```

Return Value

Value of "Font" property of the object assigned to the class instance. If there is no object assigned, it returns "".

Font (Set Method)

Sets new value for "Font" property.

```
bool  Font(
   string  font        // property value
   )
```

Parameters

font

[in] New value for "Font" property.

Return Value

true - success, false - cannot change the property.

FontSize (Get Method)

Gets the value of "Font size" property.

```
int  FontSize() const
```

Return Value

Value of "FontSize" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

FontSize (Set Method)

Sets new value for "Font size" property.

```
bool  FontSize(
   int  size      // property value
   )
```

Parameters

size

[in] New value for "Font size" property.

Return Value

true - successful, false - cannot change the property.

Anchor (Get Method)

Gets the value of "Anchor" property.

```
ENUM_ANCHOR_POINT  Anchor() const
```

Return Value

Value of "Anchor" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets new value for "Anchor" property.

```
bool  Anchor(
   ENUM_ANCHOR_POINT  anchor      // property value
   )
```

Parameters

anchor

[in] New value for "Anchor" property.

Return Value

true - success, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by the FileOpen function.

Return Value

true - success, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by the FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_TEXT for CChartObjectText).

CChartObjectLabel

CChartObjectLabel is a class for simplified access to "Label" graphical object properties.

Description

CChartObjectLabel class provides access to "Label" object properties.

Declaration

```
   class CChartObjectLabel : public CChartObjectText
```

Title

```
   #include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectText
      - CChartObjectLabel

Direct descendants

CChartObjectEdit, CChartObjectRectLabel

Class Methods by Groups

Create	
Create	Creates "Label" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
Corner	Gets/sets "Corner" property
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

See also

Object types, Object properties, Chart angle, Methods of Object Binding, Graphic objects

Create

Creates "Label" graphical object.

```
bool  Create(
   long     chart_id,      // chart identifier
   string   name,          // object name
   int      window,        // chart window
   int      X,             // X coordinate
   int      Y              // Y coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X       // property value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y       // property value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

X_Size

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER  Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // property value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time       // any value
   )
```

Parameters

time

[in] Any value of datetime type.

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price       // any value
   )
```

Parameters

price

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_LABEL for CChartObjectLabel).

CChartObjectEdit

CChartObjectEdit is a class for simplified access to "Edit" graphical object properties.

Description

CChartObjectEdit class provides access to "Edit" object properties.

Declaration

```
   class CChartObjectEdit : public CChartObjectLabel
```

Title

```
   #include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectText
      - CChartObjectLabel
        - CChartObjectEdit

Direct descendants

- CChartObjectButton

Class Methods by Groups

Create	
Create	Creates "Edit" graphical object
Properties	
TextAlign	Gets/sets "TextAlign" property
X_Size	Gets "X Size" property
Y_Size	Gets "Y Size" property
BackColor	Gets/sets "Background Color" property
BorderColor	Gets/sets "Border Color" property
ReadOnly	Gets/sets "Read Only" property
Angle	Gets/sets "Angle" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file

Create	
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

Methods inherited from class CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

See also

Object types, Object properties, Chart angle, Methods of Object Binding, Graphic objects

Create

Creates "Edit" graphical object.

```
bool  Create(
   long     chart_id,      // chart identifier
   string   name,          // object name
   int      window,        // chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // X size
   int      sizeY          // Y size
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Return Value

true - successful, false - error.

TextAlign (Get method)

Gets the value of "TextAlign" property (text alignment mode).

```
ENUM_ALIGN_MODE  TextAlign() const
```

Return Value

Value of "TextAlign" property of the object assigned to the class instance.

TextAlign (Set method)

Sets the value of "TextAlign" property (text aligment mode).

```
bool  TextAlign(
   ENUM_ALIGN_MODE  align      // property value
   )
```

Parameters

align

[in] New value of "TextAlign" property.

Return Value

true - success, false - cannot change the property.

X_Size

Sets the value for "X_Size" property.

```
bool  X_Size(
   int  size      // property value
   )
```

Parameters

size

[in] New value for "X_Size" property.

Return Value

true - success, false - cannot change the property.

Y_Size

Sets the value for "Y_Size" property.

```
bool  Y_Size(
   int  size       // property value
   )
```

Parameters

size

[in] New value for "Y_Size" property.

Return Value

true - success, false - cannot change the property.

BackColor (Get Method)

Gets the value of "BackColor" property.

```
color  BackColor() const
```

Return Value

Value of "BackColor" property of the object assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BackColor (Set Method)

Sets the value for "BackColor" property.

```
bool  BackColor(
   color  new_color      // property value
   )
```

Parameters

new_color

[in] New value for "BackColor" property.

Return Value

true - success, false - cannot change the property.

BorderColor (Get Method)

Gets the value of "Border Color" property.

```
color  BorderColor() const
```

Return Value

Value of "Border Color" property of the object assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BorderColor (Set Method)

Sets new value for "Border Color" property.

```
bool  BorderColor(
   color  new_color      // new property value
   )
```

Parameters

new_color

[in] New value for "Border Color" property.

Return Value

true - success, false - cannot change the property.

ReadOnly (Get Method)

Gets the value of "Read Only" property.

```
bool  ReadOnly() const
```

Return Value

Value of "Read Only" property of the object assigned to the class instance. If there is no object assigned, it returns false.

ReadOnly (Set Method)

Sets the value for "Read Only" property.

```
bool  ReadOnly(
   const  bool   flag      // new property value
   )
```

Parameters

flag

[in] New value for "Read Only" property (true means text editing is disabled).

Return Value

true - success, false - cannot change the property.

Angle

Prohibits changes of the "Angle" property.

```
bool  Angle(
   double  angle       // any value
   )
```

Parameters

angle

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - success, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - success, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_EDIT for CChartObjectEdit).

CChartObjectButton

CChartObjectButton is a class for simplified access to "Button" graphical object properties.

Description

CChartObjectButton class provides access to "Button" object properties.

Declaration

```
class CChartObjectButton : public CChartObjectEdit
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectText
      - CChartObjectLabel
        - CChartObjectEdit
          - CChartObjectButton

Direct descendants

CChartObjectPanel

Class Methods by Groups

Create	
Create	Inherited form CChartObjectEdit class
Properties	
State	Gets/sets button state (Pressed/Depressed)
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

Methods inherited from class CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

Methods inherited from class CChartObjectEdit

X_Size, Y_Size, BackColor, BackColor, BorderColor, BorderColor, ReadOnly, ReadOnly, TextAlign, TextAlign, Angle, Create

See also

Object types, Object properties, Chart angle, Methods of object binding, Graphic objects

State (Get Method)

Gets the value of "State" property.

```
bool  State() const
```

Return Value

Value of "State" property of the object assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool  State(
   bool  state      // property value
   )
```

Parameters

X

[in] New value for "State" property.

Return Value

true - success, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_BUTTON for CChartObjectButton).

CChartObjectSubChart

CChartObjectSubChart is a class for simplified access to "Chart" graphical object properties.

Description

CChartObjectSubChart class provides access to "Chart" object properties.

Declaration

```
class CChartObjectSubChart : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectSubChart.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectSubChart

Class Methods by Groups

Create	
Create	Creates "Chart" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
Corner	Gets/sets "Corner" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
Symbol	Gets/sets "Symbol" property
Period	Gets/sets "Period" property
Scale	Gets/sets "Scale" property
DateScale	Gets/sets "Show date scale" property
PriceScale	Gets/sets "Show price scale" property
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file

Create	
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Object properties, Chart angle, Graphic objects

Create

Creates "SubChart" graphical object.

```
bool  Create(
   long     chart_id,      // chart identifier
   string   name,          // object name
   int      window,        // chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // X size
   int      sizeY          // Y size
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X       // property value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y      // property value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // property value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

X_Size (Get Method)

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Size (Set Method)

Sets new value for "X_Size" property.

```
bool  X_Size(
   int  X      // property value
   )
```

Parameters

X

[in] New value for "X_Size" property.

Return Value

true - successful, false - cannot change the property.

Y_Size (Get Method)

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size (Set Method)

Sets new value for "Y_Size" property.

```
bool  Y_Size(
   int  Y       // property value
   )
```

Parameters

Y

[in] New value for "Y_Size" property.

Return Value

true - successful, false - cannot change the property.

Symbol (Get Method)

Gets the value of "Symbol" property.

```
string  Symbol() const
```

Return Value

Value of "Symbol" property of the object assigned to the class instance. If there is no object assigned, it returns "".

Symbol (Set Method)

Sets new value for "Symbol" property.

```
bool  Symbol(
   string  symbol      // symbol
   )
```

Parameters

symbol

[in] New value for "Symbol" property.

Return Value

true - successful, false - cannot change the property.

Period (Get Method)

Gets the value of "Period" property.

```
int  Period() const
```

Return Value

Value of "Period" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Period (Set Method)

Sets new value for "Period" property.

```
bool  Period(
   int  period      // period
   )
```

Parameters

period

[in] New value for "Period" property.

Return Value

true - successful, false - cannot change the property.

Scale (Get Method)

Gets the value of "Scale" property.

```
double  Scale() const
```

Return Value

Value of "Scale" property of the object assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   double  scale       // property value
   )
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

DateScale (Get Method)

Gets the value of "DateScale" flag.

```
bool  DateScale() const
```

Return Value

Value of "DateScale" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

DateScale (Set Method)

Sets new value for "DateScale" property.

```
bool  DateScale(
   bool  scale      // flag value
   )
```

Parameters

scale

[in] New value for "DateScale" flag.

Return Value

true - successful, false - cannot change the flag.

PriceScale (Get Method)

Gets the value of "PriceScale" flag.

```
bool  PriceScale() const
```

Return Value

Value of "PriceScale" flag of the object assigned to the class instance. If there is no object assigned, it returns false.

PriceScale (Set Method)

Sets new value for "PriceScale" flag.

```
bool  PriceScale(
   bool  scale      // flag value
   )
```

Parameters

scale

[in] New value for "PriceScale" flag.

Return Value

true - successful, false - cannot change the flag.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time      // any value
   )
```

Parameters

time

[in]

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price      // any value
   )
```

Parameters

price

[in]

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_CHART for CChartObjectSubChart).

CChartObjectBitmap

CChartObjectBitmap is a class for simplified access to "Bitmap" graphical object properties.

Description

CChartObjectBitmap class provides access to "Bitmap" object properties.

Declaration

```
class CChartObjectBitmap : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Inheritance hierarchy

CObject
    CChartObject
        CChartObjectBitmap

Class Methods by Groups

Create	
Create	Creates "Bitmap" graphical object
Properties	
BmpFile	Gets/sets "BMP Filename" property
X_Offset	Gets/sets "X_Offset" property
Y_Offset	Gets/sets "Y_Offset" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Object properties, Graphic objects

Create

Creates "Bitmap" graphical object.

```
bool  Create(
   long       chart_id,      // chart identifier
   string     name,          // object name
   int        window,        // chart window
   datetime   time,          // time coordinate
   double     price          // price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Return Value

true - successful, false - error.

BmpFile (Get Method)

Gets the value of "BmpFile" property.

```
string  BmpFile() const
```

Return Value

Value of "BmpFile" property of the object assigned to the class instance. If there is no object assigned, it returns false.

BmpFile (Set Method)

Sets new value for "BmpFile" property.

```
bool  BmpFile(
   string  name      // property value
   )
```

Parameters

name

[in] New value for "BmpFile" property.

Return Value

true - successful, false - cannot change the property.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the upper left corner) of the CChartObjectBitmap graphical object.

```
int  X_Offset() const
```

Return Value

Value of "X_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the upper left corner) of the CChartObjectBitmap graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool  X_Offset(
   int  X      // property value
   )
```

Parameters

X

[in] New value for "X_Offset" property.

Return Value

true - successful, false - cannot change the property.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the upper left corner) of the CChartObjectBitmap graphical object.

```
int  Y_Offset() const
```

Return Value

Value of "Y_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "Y_Offset" property (the upper left corner) of the CChartObjectBitmap graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool  Y_Offset(
   int   Y      // property value
   )
```

Parameters

Y

[in] New value for "Y_Offset" property.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_BITMAP for CChartObjectBitmap).

CChartObjectBmpLabel

CChartObjectBmpLabel is a class for simplified access to "Bitmap Label" graphical object properties.

Description

CChartObjectBmpLabel class provides access to "Bitmap Label" object properties.

Declaration

```
class CChartObjectBmpLabel : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Inheritance hierarchy

CObject
  CChartObject
    CChartObjectBmpLabel

Class Methods by Groups

Create	
Create	Creates "BmpLabel" graphical object
Properties	
X_Distance	Gets/sets "X_Distance" property
Y_Distance	Gets/sets "Y_Distance" property
X_Offset	Gets/sets "X_Offset" property
Y_Offset	Gets/sets "Y_Offset" property
Corner	Gets/sets "Corner" property
X_Size	Gets/sets "X_Size" property
Y_Size	Gets/sets "Y_Size" property
BmpFileOn	Gets/sets "BmpFileOn" property for button pressed state (On)
BmpFileOff	Gets/sets "BmpFileOff" property for button depressed state (Off)
State	Gets/sets "Button State" property (Pressed/Depressed)
Time	"Stub" for time coordinate change
Price	"Stub" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file

Create	
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

See also

Object types, Object properties, Chart angle, Graphic objects

Create

Creates "BmpLabel" graphical object.

```
bool  Create(
   long     chart_id,      // chart identifier
   string   name,          // object name
   int      window,        // chart window
   int      X,             // X coordinate
   int      Y              // Y coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 – current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 – main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Return Value

true - successful, false - error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Return Value

Value of "X_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X      // property value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Return Value

true - successful, false - cannot change the property.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Return Value

Value of "Y_Distance" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y       // property value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Return Value

true - successful, false - cannot change the property.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the upper left corner) of the CCartObjectBmpLabel graphical object.

```
int  X_Offset() const
```

Return Value

Value of "X_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the upper left corner) of the CChartObjectBitmap graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool  X_Offset(
   int  X      // property value
   )
```

Parameters

X

[in] New value for "X_Offset" property.

Return Value

true - successful, false - cannot change the property.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the upper left corner) of the CCartObjectBmpLabel graphical object.

```
int  Y_Offset() const
```

Return Value

Value of "Y_Offset" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "Y_Offset" property (the upper left corner) of the CCartObjectBmpLabel graphical object. The value is set in pixels relative to the upper left corner of the original image.

```
bool  Y_Offset(
   int  Y       // property value
   )
```

Parameters

Y

[in] New value for "Y_Offset" property.

Return Value

true - successful, false - cannot change the property.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Return Value

Value of "Corner" property of the object assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // property value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Return Value

true - successful, false - cannot change the property.

X_Size

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Return Value

Value of "X_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Return Value

Value of "Y_Size" property of the object assigned to the class instance. If there is no object assigned, it returns 0.

BmpFileOn (Get Method)

Gets the value of "BmpFileOn" property.

```
string  BmpFileOn() const
```

Return Value

Value of "BmpFileOn" property of the object assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOn (Set Method)

Sets new value for "BmpFileOn" property.

```
bool  BmpFileOn(
   string  name       // file name
   )
```

Parameters

name

[in] New value for "BmpFileOn" property.

Return Value

true - successful, false - cannot change the property.

BmpFileOff (Get Method)

Gets the value of "BmpFileOff" property.

```
string  BmpFileOff() const
```

Return Value

Value of "BmpFileOff" property of the object assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOff (Set Method)

Sets new value for "BmpFileOff" property.

```
bool  BmpFileOff(
   string  name        // file name
   )
```

Parameters

name

[in] New value for "BmpFileOff" property.

Return Value

true - successful, false - cannot change the property.

State (Get Method)

Gets the value of "State" property.

```
bool  State() const
```

Return Value

Value of "State" property of the object assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool  State(
   bool  state       // property value
   )
```

Parameters

state

[in] New value for "State" property.

Return Value

true - successful, false - cannot change the property.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time      // any value
   )
```

Parameters

time

[in] Any value of datetime type.

Return Value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price       // any value
   )
```

Parameters

price

[in] Any value of double type.

Return Value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_BITMAP_LABEL for CChartObjectBmpLabel).

CChartObjectRectLabel

CChartObjectRectLabel is a class for simplified access to "Rectangle Label" graphical object properties.

Description

CChartObjectRectLabel class provides access to "Rectangle Label" object properties.

Declaration

```
   class CChartObjectRectLabel : public CChartObjectLabel
```

Title

```
   #include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Inheritance hierarchy

- CObject
  - CChartObject
    - CChartObjectText
      - CChartObjectLabel
        - CChartObjectRectLabel

Class Methods by Groups

Create	
Create	Creates "RectLabel" graphical object
Properties	
X_Size	Sets the horizontal size
Y_Size	Sets the vertical size
BackColor	Gets/sets the background color
Angle	Prohibits slope angle change
BorderType	Gets/sets type of the border
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CChartObject

ChartId, Window, Name, Name, NumPoints, Attach, SetPoint, Delete, Detach, Time, Time, Price, Price, Color, Color, Style, Style, Width, Width, Background, Background, Fill, Fill, Z_Order, Z_Order, Selected, Selected, Selectable, Selectable, Description, Description, Tooltip, Tooltip, Timeframes, Timeframes, CreateTime, LevelsCount, LevelsCount, LevelColor, LevelColor, LevelStyle, LevelStyle, LevelWidth, LevelWidth, LevelValue, LevelValue, LevelDescription, LevelDescription, GetInteger, GetInteger, SetInteger, SetInteger, GetDouble, GetDouble, SetDouble, SetDouble, GetString, GetString, SetString, SetString, ShiftObject, ShiftPoint

Methods inherited from class CChartObjectText

Angle, Angle, Font, Font, FontSize, FontSize, Anchor, Anchor, Create

Methods inherited from class CChartObjectLabel

X_Distance, X_Distance, Y_Distance, Y_Distance, X_Size, Y_Size, Corner, Corner, Time, Price, Create

See also

Object types, Object properties, Graphic objects

Create

Creates the "CChartObjectRectLabel" graphic object.

```
bool  Create(
   long     chart_id,      // chart ID
   string   name,          // object name
   int      window,        // chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // horizontal size
   int      sizeY          // vertical size
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - main window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] Horizontal size.

sizeY

[in] Vertical size.

Return Value

true - successful, false - error.

X_Size

Sets the value of "X_Size" property.

```
bool  X_Size(
   int   size      // property value
   )
```

Parameters

size

[in] New horizontal size property value.

Return Value

true - successful, false - cannot change the property.

Note

To get the values of "X_Size" and "Y_Size" properties, use the X_Size and Y_Size methods of the parent CChartObjectLabel class.

Y_Size

Sets the value of "Y_Size" property.

```
bool  Y_Size(
   int  size       // property value
   )
```

Parameters

size

[in] New vertical size property value.

Return Value

true - successful, false - cannot change the property.

Note

To get the values of "X_Size" and "Y_Size" properties, use the X_Size and Y_Size methods of the parent CChartObjectLabel class.

BackColor

Gets the background color property value.

```
color  BackColor() const
```

Return Value

Background color property value of the object assigned to the class instance. If there is no object assigned, it returns 0.

BackColor

Sets the background color property value.

```
bool  BackColor(
   color  new_color      // property value
   )
```

Parameters

new_color

[in] New background color property value.

Return Value

true - successful, false - cannot change the property.

Angle

Prohibits changing the slope angle property.

```
bool  Angle(
   double  angle      // any value
   )
```

Parameters

angle

[in] Any value of double type.

Return Value

Always false.

BorderType

Gets border type property value.

```
int  BorderType() const
```

Return Value

Border type property value of the object assigned to the class instance. If there is no object assigned, it returns 0.

BorderType

Sets border type property value.

```
bool  BorderType(
   int  type       // property value
   )
```

Parameters

type

[in] New border type property value.

Return Value

true - successful, false - cannot change the property.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Return Value

true - successful, false - error.

Type

Returns graphical object type identifier.

```
virtual int  Type() const
```

Return Value

Object type identifier (OBJ_RECTANGLE_LABEL for CChartObjectRectangleLabel).

Custom graphics

This section provides tools for working with custom graphics.

Their use greatly facilitates plotting custom charts, drawings and visualization of data.

There are individual classes for creating graphical objects and primitives, for drawing various types of pie charts and curves. Various options for displaying objects are implemented: changing the style and color of lines, filling, working with series of data on the chart, etc.

Class	Description
CCanvas	Class for simplified creation of custom images
CChartCanvas	Base class for implementing classes intended for drawing charts and their elements
CHistogramChart	Class for plotting histograms
CLineChart	Class for plotting curves
CPieChart	Class for plotting pie charts

CCanvas

CCanvas is a class for simplified creation of custom images.

Description

CCanvas provides creation of a graphical resource (with or without binding to a chart object) and drawing graphic primitives.

Declaration

```
class CCanvas
```

Title

```
#include <Canvas\Canvas.mqh>
```

Inheritance hierarchy

CCanvas

Direct descendants

CChartCanvas, CFlameCanvas

Class methods by groups

Creating	
Attach	Attaches the OBJ_BITMAP_LABEL object to an instance of the CCanvas class
Create	Creates a graphical resource without binding to a chart object
CreateBitmap	Create a graphical resource bound to a chart object
CreateBitmapLabel	Create a graphical resource bound to a chart object
Destroy	Destroys a graphical resource
Properties	
ChartObjectName	Gets the name of a bound chart object
ResourceName	Gets the name of a graphical resource
Width	Gets the width of a graphical resource
Height	Gets the height of a graphical resource
LineStyleSet	Sets the line style
Updates an object on the screen	
Update	Displays changes on the screen

Creating	
Resize	Resizes a graphical resource
Erasing/Filling with color	
Erase	Erases or fills with the specified color
Data access	
PixelGet	Gets a color of the dot with the specified coordinates
PixelSet	Sets color of the dot with the specified coordinates
Draws primitives	
LineVertical	Draws a vertical line
LineHorizontal	Draws a horizontal line
Line	Draws a freehand line
Polyline	Draws a polyline
Polygon	Draws a polygon
Rectangle	Draws a rectangle
Circle	Draws a circle
Triangle	Draws a triangle
Ellipse	Draws an ellipse
Arc	Draws an ellipse arc
Pie	Draws an ellipse sector
Draws filled primitives	
FillRectangle	Draws a filled rectangle
FillCircle	Draws a filled circle
FillTriangle	Draws a filled triangle
FillPolygon	Draws a filled polygon
FillEllipse	Draws a filled ellipse
Fill	Fills an area
Draws primitives with antialiasing	
PixelSetAA	Draws a pixel
LineAA	Draws a line
PolylineAA	Draws a polyline
PolygonAA	Draws a polygon

Creating	
TriangleAA	Draws a triangle
CircleAA	Draws a circle
EllipseAA	Draws an ellipse
LineWu	Draws a line
PolylineWu	Draws a polyline
PolygonWu	Draws a polygon
TriangleWu	Draws a triangle
CircleWu	Draws a circle
EllipseWu	Draws an ellipse
LineThick	Draws a segment of a freehand line having a specified width using antialiasing algorithm.
LineThickVertical	Draws a vertical segment of a freehand line having a specified width using antialiasing algorithm.
LineThickHorizontal	Draws a horizontal segment of a freehand line having a specified width using antialiasing algorithm.
PolygonSmooth	Draws a polygon with a specified width using two antialiasing algorithms
PolygonThick	Draws a polygon with a specified width using antialiasing algorithm
PolylineSmooth	Draws a polyline with a specified width using two antialiasing algorithms
PolylineThick	Draws a polyline with a specified width using antialiasing algorithm
Text	
FontSet	Sets font parameters
FontNameSet	Sets font name
FontSizeSet	Sets font size
FontFlagsSet	Sets font flags
FontAngleSet	Sets font slope angle
FontGet	Gets font parameters
FontNameGet	Gets font name
FontSizeGet	Gets font size
FontFlagsGet	Gets font flags
FontAngleGet	Gets font slope angle

Creating	
TextOut	Displays text
TextWidth	Gets the text width
TextHeight	Gets the text height
TextSize	Gets the text size
Transparency	
TransparentLevelSet	Sets transparency level
Input/output	
LoadFromFile	Reads an image from a BMP file

Attach

Gets the graphical resource from an OBJ_BITMAP_LABEL object and attaches it to an instance of the CCanvas class.

```
bool  Attach(
   const long         chart_id,                              // chart identifier
   const string       objname,                               // object name
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // color processing metho
```

Creates a graphical resource for an OBJ_BITMAP_LABEL object and attaches it to an instance of the CCanvas class.

```
bool  Attach(
   const long         chart_id,                              // chart identifier
   const string       objname,                               // object name
   const int          width,                                 // image width in pixels
   const int          height,                                // image height in pixels
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // color processing metho
```

Parameters

chart_id

[out] Chart identifier.

objname

[in] Name of the graphical object.

width

[in] Image width in the resource.

height

[in] Image height in the resource.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Alpha channel processing method. The alpha channel is ignored by default.

Return Value

true - if successful, false - if failed to attach the object.

Arc

Draws an arc of an ellipse inscribed in a rectangle with corners at (x1,y1) and (x2,y2). The arc boundaries are clipped by lines from the center of the ellipse, which extend to two points with coordinates (x3,y3) and (x4,y4).

```
void  Arc(
   int         x1,      // X coordinate of the upper left corner of the rectangle
   int         y1,      // Y coordinate of the upper left corner of the rectangle
   int         x2,      // X coordinate of the bottom right corner of the rectangle
   int         y2,      // Y coordinate of the bottom right corner of the rectangle
   int         x3,      // X coordinate of the first point to find the arc boundaries
   int         y3,      // Y coordinate of the first point to find the arc boundaries
   int         x4,      // X coordinate of the second point to find the arc boundaries
   int         y4,      // Y coordinate of the second point to find the arc boundaries
   const uint  clr      // color
   );
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

x3

[in] X coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y3

[in] Y coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

x4

[in] X coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y4

[in] Y coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

clr

[in] Color in ARGB format. Use the ColorToARGB() function to convert a color into the ARGB format.

Draws an arc of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry. The arc boundaries are cropped from the ellipse center using rays formed by angles fi3 and fi4.

```
void  Arc(
   int          x,        // X coordinate of the ellipse center
   int          y,        // Y coordinate of the ellipse center
   int          rx,       // ellipse radius on the X axis
   int          ry,       // ellipse radius on the Y axis
   int          fi3,      // angle of ray from ellipse center, which defines the first
   int          fi4,      // angle of ray from ellipse center, which defines the second
   const uint   clr       // color
   );
```

Draws an arc of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry, and also returns the coordinates of the arc boundaries. The arc boundaries are cropped from the ellipse center using rays formed by angles fi3 and fi4.

```
void  Arc(
   int          x,        // X coordinate of the ellipse center
   int          y,        // Y coordinate of the ellipse center
   int          rx,       // ellipse radius on the X axis
   int          ry,       // ellipse radius on the Y axis
   int          fi3,      // angle of ray from ellipse center, which defines the first
   int          fi4,      // angle of ray from ellipse center, which defines the second
   int&         x3,       // X coordinate of the first arc boundary
   int&         y3,       // Y coordinate of the first arc boundary
   int&         x4,       // X coordinate of the second arc boundary
   int&         y4,       // Y coordinate of the second arc boundary
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of the ellipse center.

y

[in] Y coordinate of the ellipse center.

rx

[in] Ellipse radius on the X axis, in pixels.

ry

[in] Ellipse radius on the Y axis, in pixels.

fi3

[in] Angle in radians, which defines the first boundary of the arc.

fi4

[in] Angle in radians, which defines the second boundary of the arc.

x3

[out] Variable to get the X coordinate of the first arc boundary.

y3

[out] Variable to get the Y coordinate of the first arc boundary.

x4

[out] Variable to get the X coordinate of the second arc boundary.

y4

[out] Variable to get the Y coordinate of the second arc boundary.

clr

[in] Color in ARGB format. Use the ColorToARGB() function to convert a color into the ARGB format.

Examples of calling the class methods:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int      Width=600;
   int      Height=400;
//--- create canvas
   if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
     {
      Print("Error creating canvas: ",GetLastError());
     }
//--- clear canvas
   canvas.Erase(clrWhite);
//--- draw rectangle
   canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- draw first arc
   canvas.Arc(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrRed));
   int x1,y1,x2,y2;
//--- draw second arc
   canvas.Arc(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,x1,y1,x2,y2,ColorToARGB(cl
//--- print coordinates of arc
   PrintFormat("First point of arc at (%G,%G), second point of arc at (%G,%G)",x1,y1,
   canvas.CircleAA(x1,y1,3, ColorToARGB(clrRed));
   canvas.CircleAA(x2,y2,3, ColorToARGB(clrBlue));
//--- show updated canvas
   canvas.Update();
  }
```

Pie

Draws a filled sector of an ellipse inscribed in a rectangle with corners at (x1,y1) and (x2,y2). The sector boundaries are clipped by lines from the center of the ellipse, which extend to two points with coordinates (x3,y3) and (x4,y4).

```
void  Pie(
   int          x1,      // X coordinate of the upper left corner of the rectangle
   int          y1,      // Y coordinate of the upper left corner of the rectangle
   int          x2,      // X coordinate of the bottom right corner of the rectangle
   int          y2,      // Y coordinate of the bottom right corner of the rectangle
   int          x3,      // X coordinate of the first point to find the arc boundaries
   int          y3,      // Y coordinate of the first point to find the arc boundaries
   int          x4,      // X coordinate of the second point to find the arc boundaries
   int          y4,      // Y coordinate of the second point to find the arc boundaries
   const uint   clr,     // line color
   const uint   fill_clr // fill color
   );
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

x3

[in] X coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y3

[in] Y coordinate of the first point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

x4

[in] X coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

y4

[in] Y coordinate of the second point, to which a line from the rectangle center is drawn in order to obtain the arc boundary.

clr

[in] Border color of the sector in the ARGB format.

fill_clr

[in] Fill color of the sector in the ARGB format. Use the ColorToARGB() function to convert a color into the ARGB format.

Draws a filled sector of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry. The sector boundaries are cropped from the ellipse center by rays formed by angles fi3 and fi4.

```
void  Pie(
   int          x,        // X coordinate of the ellipse center
   int          y,        // Y coordinate of the ellipse center
   int          rx,       // ellipse radius on the X axis
   int          ry,       // ellipse radius on the Y axis
   int          fi3,      // angle of ray from ellipse center, which defines the first 
   int          fi4,      // angle of ray from ellipse center, which defines the second
   const uint   clr,      // line color
   const uint   fill_clr // fill color
   );
```

Draws a filled sector of an ellipse with center at point (x,y), inscribed in rectangle, with radii rx and ry, and also returns the coordinates of the arc boundaries. The sector boundaries are cropped from the ellipse center by rays formed by angles fi3 and fi4.

```
void  Pie(
   int          x,        // X coordinate of the ellipse center
   int          y,        // Y coordinate of the ellipse center
   int          rx,       // ellipse radius on the X axis
   int          ry,       // ellipse radius on the Y axis
   int          fi3,      // angle of ray from ellipse center, which defines the first 
   int          fi4,      // angle of ray from ellipse center, which defines the second
   int&         x3,       // X coordinate of the first arc boundary
   int&         y3,       // Y coordinate of the first arc boundary
   int&         x4,       // X coordinate of the second arc boundary
   int&         y4,       // Y coordinate of the second arc boundary
   const uint   clr,      // line color
   const uint   fill_clr // fill color
   );
```

Parameters

x

[in] X coordinate of the ellipse center.

y

[in] Y coordinate of the ellipse center.

rx

[in] Ellipse radius on the X axis, in pixels.

ry

[in] Ellipse radius on the X axis, in pixels.

fi3

[in] Angle in radians, which defines the first boundary of the arc.

fi4

[in] Angle in radians, which defines the second boundary of the arc.

x3

[out] Variable to get the X coordinate of the first arc boundary.

y3

[out] Variable to get the Y coordinate of the first arc boundary.

x4

[out] Variable to get the X coordinate of the second arc boundary.

y4

[out] Variable to get the Y coordinate of the second arc boundary.

clr

[in] Border color of the sector in the ARGB format.

fill_clr

[in] Fill color of the sector in the ARGB format. Use the ColorToARGB() function to convert a color into the ARGB format.

Examples of calling the class methods:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int      Width=600;
   int      Height=400;
//--- create canvas
   if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
     {
      Print("Error creating canvas: ",GetLastError());
     }
//--- clear canvas
   canvas.Erase(clrWhite);
//--- draw rectangle
   canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- draw first pie
   canvas.Pie(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrBlue),ColorToARGB(
//--- draw second pie
   canvas.Pie(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,ColorToARGB(clrGreen),Colo
//--- show updated canvas
   canvas.Update();
```

```
    DebugBreak();
  }
```

FillPolygon

Draws a filled polygon.

```
void  FillPolygon(
   int&          x,       // array with the X coordinates of polygon points
   int&          y,       // array with the Y coordinates of polygon points
   const uint   clr       // color
   );
```

Parameters

x

[in] Array of the X coordinates of the polygon points.

y

[in] Array of the Y coordinates of the polygon points.

clr

[in] Color in ARGB format.

FillEllipse

Draws a filled ellipse inscribed in a rectangle with the specified coordinates.

```
void  FillPolygon(
   int          x1,       // X coordinate of the upper left corner of the rectangle
   int          y1,       // Y coordinate of the upper left corner of the rectangle
   int          x2,       // X coordinate of the bottom right corner of the rectangle
   int          y2,       // Y coordinate of the bottom right corner of the rectangle
   const uint   clr        // ellipse color
   );
```

Parameters

x1

[in] X coordinate of the top left corner forming the rectangle.

y1

[in] Y coordinate of the top left corner forming the rectangle.

x2

[in] X coordinate of the bottom right corner forming the rectangle.

y2

[in] Y coordinate of the bottom right corner forming the rectangle.

clr

[in] Color in ARGB format.

GetDefaultColor

Returns a predefined color by its index.

```
static uint  GetDefaultColor(
   const uint  i       // index
   );
```

Parameters

i

[in] Index to get the color.

Return Value

Color.

ChartObjectName

Receives the name of a bound chart object.

```
string  ChartObjectName();
```

Return Value

the name of a bound chart object

Circle

Draws a circle

```
void  Circle(
   int          x,        // X coordinate
   int          y,        // Y coordinate
   int          r,        // radius
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

CircleAA

Draws a circle using antialiasing algorithm

```
void  CircleAA(
   const int      x,         // X coordinate
   const int      y,         // Y coordinate
   const double   r,         // radius
   const uint     clr        // color
   );
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

CircleWu

Draws a circle using Wu's anti-aliasing algorithm

```
void  CircleWu(
   const int      x,         // X coordinate
   const int      y,         // Y coordinate
   const double   r,         // radius
   const uint     clr        // color
   );
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

Create

Creates a graphical resource without binding to a chart object.

```
virtual bool  Create(
   const string        name,                                  // name
   const int           width,                                 // width
   const int           height,                                // height
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // format
   );
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

CreateBitmap

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool  CreateBitmap(
   const string        name,                                    // name
   const datetime      time,                                    // time
   const double        price,                                   // price
   const int           width,                                   // width
   const int           height,                                  // height
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA         // format
   );
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool  CreateBitmap(
   const long          chart_id,                                // chart ID
   const int           subwin,                                  // subwindow number
   const string        name,                                    // name
   const datetime      time,                                    // time
   const double        price,                                   // price
   const int           width,                                   // width
   const int           height,                                  // height
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA         // format
   );
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

time

[in] Chart object anchor point time coordinate.

price

[in] Chart object anchor point price coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

CreateBitmapLabel

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool  CreateBitmapLabel(
   const string        name,                                  // name
   const int           x,                                     // X coordinate
   const int           y,                                     // Y coordinate
   const int           width,                                 // width
   const int           height,                                // height
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // format
   );
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool  CreateBitmapLabel(
   const long          chart_id,                              // chart ID
   const int           subwin,                                // subwindow number
   const string        name,                                  // name
   const int           x,                                     // X coordinate
   const int           y,                                     // Y coordinate
   const int           width,                                 // width
   const int           height,                                // height
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // format
   );
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

x

[in] Chart object anchor point X coordinate.

y

[in] Chart object anchor point Y coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Return Value

true - successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

Destroy

Destroys a graphical resource.

```
void  Destroy();
```

Note

If a graphical resource has been bound to a chart object, the latter is deleted.

Ellipse

Draws an ellipse using two points.

```
void  Ellipse(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

EllipseAA

Draws an ellipse based on two points using anti-aliasing algorithm.

```
void  EllipseAA(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

EllipseWu

Draws an ellipse based on two points using Wu's anti-aliasing algorithm.

```
void  EllipseWu(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming an ellipse.

y1

[in] Y coordinate of the first point forming an ellipse.

x2

[in] X coordinate of the second point forming an ellipse.

y2

[in] Y coordinate of the second point forming an ellipse.

clr

[in] Color in ARGB format.

Erase

Erases or fills with the specified color.

```
void  Erase(
   const uint   clr=0       // color
   );
```

Parameters

clr=0

[in] Color in ARGB format.

Fill

Fills an area.

```
void  Fill(
   int          x,       // X coordinate
   int          y,       // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x

[in] X coordinate of filling starting point.

y

[in] Y coordinate of filling starting point.

clr

[in] Color in ARGB format.

FillCircle

Draws a filled circle.

```
void  FillCircle(
   int          x,        // X coordinate
   int          y,        // Y coordinate
   int          r,        // radius
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of a filled circle center.

y

[in] Y coordinate of a filled circle center.

r

[in] Filled circle radius.

clr

[in] Color in ARGB format.

FillRectangle

Draws a filled rectangle.

```
void  FillRectangle(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

FillTriangle

Draws a filled triangle.

```
void  FillTriangle(
   int           x1,       // X coordinate
   int           y1,       // Y coordinate
   int           x2,       // X coordinate
   int           y2,       // Y coordinate
   int           x3,       // X coordinate
   int           y3,       // Y coordinate
   const uint    clr       // color
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

FontAngleGet

Receives font slope angle.

```
uint  FontAngleGet();
```

Return Value

font slope angle

FontAngleSet

Sets font slope angle.

```
bool  FontAngleSet(
   uint  angle      // angle
   );
```

Parameters

angle

[in] Font slope angle in tenths of a degree.

Return Value

true - successful, otherwise - false

FontFlagsGet

Receives font flags.

```
uint  FontFlagsGet();
```

Return Value

font flags

FontFlagsSet

Sets font flags.

```
bool  FontFlagsSet(
   uint  flags      // flags
   );
```

Parameters

flags

[in] Font creation flags. See TextSetFont() function description to learn more about the flags.

Return Value

true - successful, otherwise - false

FontGet

Receives the current font parameters.

```
void  FontGet(
   string&  name,       // name
   int&     size,       // size
   uint&    flags,      // flags
   uint&    angle       // slope angle
   );
```

Parameters

name

[out] Reference to the variable for returning a font name.

size

[out] Reference to the variable for returning a font size.

flags

[out] Reference to the variable for returning font flags.

angle

[out] Reference to the variable for returning a font slope angle.

FontNameGet

Receives font name.

```
string  FontNameGet();
```

Return Value

font name

FontNameSet

Sets font name.

```
bool  FontNameSet(
   string  name       // name
   );
```

Parameters

name

[in] Font name. For example, "Arial".

Return Value

true - successful, otherwise - false

FontSet

Sets the current font.

```
bool  FontSet(
   const string  name,         // name
   const int     size,         // size
   const uint    flags=0,      // flags
   const uint    angle=0       // angle
   );
```

Parameters

name

[in] Font name. For example, "Arial".

size

[in] Font size. See TextSetFont() function description to learn more about setting a size.

flags=0

[in] Font creation flags. See TextSetFont() function description to learn more about the flags.

angle=0

[in] Font slope angle in tenths of a degree.

Return Value

true - successful, otherwise - false

FontSizeGet

Receives font size.

```
int  FontSizeGet();
```

Return Value

font size

FontSizeSet

Sets font size.

```
bool  FontSizeSet(
   int   size       // size
   );
```

Parameters

size

[in] Font size. See TextSetFont() function description to learn more about setting a size.

Return Value

true - successful, otherwise - false

Height

Receives the height of a graphical resource.

```
int  Height();
```

Return Value

height of a graphical resource

Line

Draws a segment of a freehand line.

```
void  Line(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineAA

Draws a segment of a freehand line using antialiasing algorithm.

```
void  LineAA(
   const int    x1,                    // X coordinate
   const int    y1,                    // Y coordinate
   const int    x2,                    // X coordinate
   const int    y2,                    // Y coordinate
   const uint   clr,                   // color
   const uint   style=UINT_MAX         // line style
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

LineWu

Draws a segment of a freehand line using Wu's anti-aliasing algorithm.

```
void  LineWu(
   const int    x1,                   // X coordinate
   const int    y1,                   // Y coordinate
   const int    x2,                   // X coordinate
   const int    y2,                   // Y coordinate
   const uint   clr,                  // color
   const uint   style=UINT_MAX        // line style
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

LineHorizontal

Draws a segment of a horizontal line.

```
void  LineHorizontal(
   int          x1,      // X coordinate
   int          x2,      // X coordinate
   int          y,       // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y

[in] Segment's Y coordinate.

clr

[in] Color in ARGB format.

LineVertical

Draws a segment of a vertical line.

```
void  LineVertical(
   int         x,        // X coordinate
   int         y1,       // Y coordinate
   int         y2,       // Y coordinate
   const uint  clr       // color
   );
```

Parameters

x

[in] Segment's X coordinate.

y1

[in] Y coordinate of the segment's first point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineStyleSet

Sets the line style.

```
void  LineStyleSet(
   const uint  style       // style
   );
```

Parameters

style

[in] Line style.

Note

The input parameter can have any of ENUM_LINE_STYLE enumeration values. Besides, it is possible to create a custom line drawing style.

LineThick

Draws a segment of a freehand line having a specified width using antialiasing algorithm.

```
void  LineThick(
   const int      x1,           // X coordinate of the segment's first point
   const int      y1,           // Y coordinate of the segment's first point
   const int      x2,           // X coordinate of the segment's second point
   const int      y2,           // Y coordinate of the segment's second point
   const uint     clr,          // color
   const int      size,         // line width
   const uint     style,        // line style
   ENUM_LINE_END  end_style     // line ends style
   )
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the ENUM_LINE_END enumeration's values

ENUM_LINE_END

ID	Description
LINE_END_ROUND	Line ends are rounded.
LINE_END_BUTT	Line ends are cut.
LINE_END_SQUARE	A line ends in a filled rectangle.

LineThickVertical

Draws a vertical segment of a freehand line having a specified width using antialiasing algorithm.

```
void  LineThickVertical(
   const int      x,             // X coordinate of the segment
   const int      y1,            // Y coordinate of the segment's first point
   const int      y2,            // Y coordinate of the segment's second point
   const uint     clr,           // color
   const int      size,          // line width
   const uint     style,         // line style
   ENUM_LINE_END  end_style      // line ends style
   )
```

Parameters

x

[in] Segment's X coordinate.

y1

[in] Y coordinate of the segment's first point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the ENUM_LINE_END enumeration's values.

LineThickHorizontal

Draws a horizontal segment of a freehand line having a specified width antialiasing.

```
void  LineThickHorizontal(
   const int       x1,           // X coordinate of the segment's first point
   const int       x2,           // X coordinate of the segment's second point
   const int       y,            // Y coordinate of the segment
   const uint      clr,          // color
   const int       size,         // line width
   const uint      style,        // line style
   ENUM_LINE_END  end_style      // line ends style
   )
```

Parameters

x1

[in] X coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y

[in] Segment's Y coordinate.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the ENUM_LINE_END enumeration's values.

LoadFromFile

Reads an image from a BMP file.

```
bool  LoadFromFile(
   const string  filename       // file name
   );
```

Parameters

filename

[in] File name (including "BMP" extension).

Return Value

true - successful, otherwise - false

PixelGet

Receives color of the point with the specified coordinates.

```
uint  PixelGet(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

Return Value

Point color in ARGB format.

PixelSet

Sets color of the point with the specified coordinates.

```
void  PixelSet(
   const int    x,         // X coordinate
   const int    y,         // Y coordinate
   const uint   clr        // color
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

PixelSetAA

Draws a point using antialiasing algorithm.

```
void  PixelSetAA(
   const double  x,        // X coordinate
   const double  y,        // Y coordinate
   const uint    clr       // color
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

Polygon

Draws a polygon.

```
void  Polygon(
   int&         x[],      // array of X coordinates
   int&         y[],      // array of Y coordinates
   const uint   clr       // color
   );
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

PolygonAA

Draws a polygon using antialiasing algorithm.

```
void  PolygonAA(
   int&        x[],              // array of X coordinates
   int&        y[],              // array of Y coordinates
   const uint  clr,              // color
   const uint  style=UINT_MAX    // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

PolygonWu

Draws a polygon using Wu's anti-aliasing algorithm.

```
void  PolygonWu(
   int&         x[],                 // array of X coordinates
   int&         y[],                 // array of Y coordinates
   const uint   clr,                 // color
   const uint   style=UINT_MAX       // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

PolygonThick

Draws a polygon with a specified width using antialiasing algorithm.

```
void  PolygonThick(
   const int&       x[],          // array with the X coordinates of polygon points
   const int&       y[],          // array with the Y coordinates of polygon points
   const uint      clr,           // color
   const int       size,          // line width
   const uint      style,         // line style
   ENUM_LINE_END   end_style      // line ends style
   )
```

Parameters

x[]

[in] Array of X coordinates of polygon points.

y[]

[in] Array of Y coordinates of polygon points.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the ENUM_LINE_END enumeration's values.

PolygonSmooth

Draws a polygon with a specified width consecutively using two antialiasing algorithms First, individual segments are smoothed based on Bezier curves. Then, the raster antialiasing algorithm is applied to the polygon built from these segments to improve the rendering quality.

```
void  PolygonSmooth(
   int&              x[],                          // array with the X coordinates of p
   int&              y[],                          // array with the Y coordinates of p
   const uint        clr,                          // color
   const int         size,                         // line width
   ENUM_LINE_STYLE   style=STYLE_SOLID,            // line style
   ENUM_LINE_END     end_style=LINE_END_ROUND,     // line ends style
   double            tension=0.5,                  // antialiasing parameter value
   double            step=10                       // length of approximation lines
   )
```

Parameters

&x[]

[in] Array of X coordinates of polygon points.

&y[]

[in] Array of Y coordinates of polygon points.

clr

[in] Color in ARGB format.

size

[in] Line width.

style=STYLE_SOLID

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style=LINE_END_ROUND

[in] Line style is one of the ENUM_LINE_END enumeration's values.

tension=0.5

[in] Smoothing parameter value.

step=10

[in] Length of approximating lines.

Polyline

Draws a polyline.

```
void  Polyline(
   int&         x[],      // array of X coordinates
   int&         y[],      // array of Y coordinates
   const uint   clr       // color
   );
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

PolylineSmooth

Draws a polyline with a specified width consecutively using two antialiasing algorithms. First, individual line segments are smoothed based on Bezier curves. Then, the raster antialiasing algorithm is applied to the polyline built from these segments to improve the rendering quality.

```
void  PolylineSmooth(
   const int&         x[],                         // array with the X coordinates of po
   const int&         y[],                         // array with the Y coordinates of po
   const uint         clr,                         // color
   const int          size,                        // line width
   ENUM_LINE_STYLE    style=STYLE_SOLID,           // line style
   ENUM_LINE_END      end_style=LINE_END_ROUND,    // line ends style
   double             tension=0.5,                 // antialiasing parameter value
   double             step=10                      // approximation step
   )
```

Parameters

&x[]

[in] Array of X coordinates of a polyline.

&y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

size

[in] Line width.

style=STYLE_SOLID

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style=LINE_END_ROUND

[in] Line style is one of the ENUM_LINE_END enumeration's values.

tension=0.5

[in] Smoothing parameter value.

step=10

[in] Approximation step.

PolylineThick

Draws a polyline with a specified width using antialiasing algorithm.

```
void  PolylineThick(
   const int       &x[],          // array with the X coordinates of polyline points
   const int       &y[],          // array with the Y coordinates of polyline points
   const uint      clr,           // color
   const int       size,          // line width
   const uint      style,         // line style
   ENUM_LINE_END   end_style      // line ends style
   )
```

Parameters

&x[]

[in] Array of X coordinates of a polyline.

&y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

size

[in] Line width.

style

[in] Line style is one of the ENUM_LINE_STYLE enumeration's values or a custom value.

end_style

[in] Line style is one of the ENUM_LINE_END enumeration's values

PolylineWu

Draws a polyline using Wu's anti-aliasing algorithm.

```
void  PolylineWu(
   int&          x[],                 // array of X coordinates
   int&          y[],                 // array of Y coordinates
   const uint    clr,                 // color
   const uint    style=UINT_MAX       // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

PolylineAA

Draws a polyline using antialiasing algorithm.

```
void  PolylineAA(
   int&         x[],              // array of X coordinates
   int&         y[],              // array of Y coordinates
   const uint   clr,              // color
   const uint   style=UINT_MAX    // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

Rectangle

Draws a rectangle using two points.

```
void  Rectangle(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

Resize

Resizes a graphical resource.

```
bool  Resize(
   const int  width,       // width
   const int  height       // height
   );
```

Parameters

width

[in] New width of a graphical resource.

height

[in] New height of a graphical resource.

Return Value

true - successful, otherwise - false

Note

When resizing, the previous image is not saved.

ResourceName

Receives the name of a graphical resource.

```
string  ResourceName();
```

Return Value

name of a graphical resource

TextHeight

Receives the text height.

```
int  TextHeight(
   const string  text       // text
   );
```

Parameters

text

[in] Text for measuring.

Return Value

text height in pixels

Note

The current font is used for measuring the text.

TextOut

Displays text.

```
void  TextOut(
   int          x,               // X coordinate
   int          y,               // Y coordinate
   string       text,            // text
   const uint   clr,             // color
   uint         alignment=0      // alignment
   );
```

Parameters

x

[in] Text anchor's X coordinate.

y

[in] Text anchor's Y coordinate.

text

[in] Text to be displayed.

clr

[in] Color in ARGB format.

alignment=0

[in] Text anchoring method. See TextOut() function description to learn more about anchoring methods.

Note

The current font is used to display the text.

TextSize

Receives the text size.

```
void  TextSize(
   const string  text,        // text
   int&          width,       // width
   int&          height       // height
   );
```

Parameters

text

[in] Text for measuring.

width

[out] Reference to the variable for returning a text width.

height

[out] Reference to the variable for returning a text height.

Note

The current font is used to measure the text.

TextWidth

Receives the text width.

```
int  TextWidth(
   const string  text      // text
   );
```

Parameters

text

[in] Text for measuring.

Return Value

text height in pixels

Note

The current font is used to measure the text.

TransparentLevelSet

Sets transparency level.

```
void  TransparentLevelSet(
   const uchar  value      // value
   );
```

Parameters

value

[in] New value of the transparency level.

Note

0 stands for full transparency, while 255 - for full opacity.

Setting a transparency level affects all that was previously drawn. The specified transparency level does not affect further constructions.

Triangle

Draws a triangle.

```
void  Triangle(
   int          x1,       // X coordinate
   int          y1,       // Y coordinate
   int          x2,       // X coordinate
   int          y2,       // Y coordinate
   int          x3,       // X coordinate
   int          y3,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

TriangleAA

Draws a triangle using antialiasing algorithm.

```
void  TriangleAA(
   const int    x1,                    // X coordinate
   const int    y1,                    // Y coordinate
   const int    x2,                    // X coordinate
   const int    y2,                    // Y coordinate
   const int    x3,                    // X coordinate
   const int    y3,                    // Y coordinate
   const uint   clr,                   // color
   const uint   style=UINT_MAX         // line style
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

TriangleWu

Draws a triangle using Wu's anti-aliasing algorithm.

```
void  TriangleWu(
   const int    x1,                 // X coordinate
   const int    y1,                 // Y coordinate
   const int    x2,                 // X coordinate
   const int    y2,                 // Y coordinate
   const int    x3,                 // X coordinate
   const int    y3,                 // Y coordinate
   const uint   clr,                // color
   const uint   style=UINT_MAX      // line style
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

Update

Displays changes on the screen.

```
void  Update(
   const bool   redraw=true      // flag
   );
```

Parameters

redraw=true

Flag of a chart redrawing necessity.

Width

Receives the width of a graphical resource.

```
int  Width();
```

Return Value

graphical resource width

CChartCanvas

Base class for implementing classes, which are used for drawing charts and their elements.

Description

This class includes methods for working with the basic elements of any chart: coordinate axes and their marks, chart legend, grid, background, etc. Here you can customize the options for displaying elements: visibility, text color, etc.

Declaration

```
class CChartCanvas : public CCanvas
```

Title

```
#include <Canvas\Charts\ChartCanvas.mqh>
```

Inheritance hierarchy

CCanvas
  CChartCanvas

Direct descendants

CHistogramChart, CLineChart, CPieChart

Class methods

Method	Action
ColorBackground	Returns and sets the background color.
ColorBorder	Returns and sets the border color.
ColorText	Returns and sets the text color.
ColorGrid	Returns and sets the grid color.
MaxData	Returns and sets the maximum amount of data (series) allowed.
MaxDescrLen	Returns and sets the maximum length of the descriptors.
ShowFlags	Returns and sets the visibility flag of the chart elements.
IsShowLegend	Returns and sets the visibility flag of the legend on the chart.
IsShowScaleLeft	Returns the visibility flag of the scale of values on the left.

Method	Action
IsShowScaleRight	Returns the visibility flag of the scale of values on the right.
IsShowScaleTop	Returns the visibility flag of the scale of values at the top.
IsShowScaleBottom	Returns the visibility flag of the scale of values at the bottom.
IsShowGrid	Returns the visibility flag of the grid on the chart.
IsShowDescriptors	Returns the visibility flag of the descriptors on the chart.
IsShowPercent	Returns the visibility flag of the percentages on the chart.
VScaleMin	Returns and sets the minimum on the vertical scale of values.
VScaleMax	Returns and sets the maximum on the vertical scale of values.
NumGrid	Returns and sets the number of vertical scale divisions when plotting the chart grid.
DataOffset	Returns and sets the data offset value.
DataTotal	Returns the total number of data series on the chart.
DrawDescriptors	Virtual method for drawing descriptors.
DrawData	Virtual method for drawing data series at the specified index.
Create	Virtual method that creates a graphical resource.
AllowedShowFlags	Sets the set of allowed visibility flags for chart elements.
ShowLegend	Sets the visibility flag for the legend.
ShowScaleLeft	Sets the visibility flag for the left scale.
ShowScaleRight	Sets the visibility flag for the right scale.
ShowScaleTop	Sets the visibility flag for the top scale.
ShowScaleBottom	Sets the visibility flag for the bottom scale.
ShowGrid	Sets the visibility flag for the grid.
ShowDescriptors	Sets the visibility flag for the descriptors.

Method	Action
ShowValue	Sets the visibility flag for the values.
ShowPercent	Sets the visibility flag for the percentages.
LegendAlignment	Sets the text alignment for the legend.
Accumulative	Sets the value accumulation flag for the series.
VScaleParams	Sets the parameters for the vertical scale of values.
DescriptorUpdate	Updates the value of the series descriptor (at the specified position).
ColorUpdate	Updates the series colors (at the specified position).
ValuesCheck	Performs internal calculations for plotting the chart.
Redraw	Redraw the chart.
DrawBackground	Draws the background.
DrawLegend	Redraws the legend.
DrawLegendVertical	Draws a vertical legend.
DrawLegendHorizontal	Draws a horizontal legend.
CalcScales	Calculates the coordinates of the scale.
DrawScales	Redraws all scales of values.
DrawScaleLeft	Redraws the left scale of values.
DrawScaleRight	Redraws the right scale of values.
DrawScaleTop	Redraws the top scale of values
DrawScaleBottom	Redraws the bottom value scale.
DrawGrid	Redraw the chart.
DrawChart	Redraw the chart.

Methods inherited from class CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

ColorBackground (Get method)

Returns the background color.

```
uint  ColorBackground()
```

Return Value

Background color.

ColorBackground (Set method)

Sets the background color.

```
void  ColorBackground(
   const uint  value,   // background color
   )
```

Parameters

value

[in] Background color.

ColorBorder (Get method)

Returns the border color.

```
uint  ColorBorder()
```

Return Value

Border color.

ColorBorder (Set method)

Sets the border color.

```
void  ColorBorder(
   const uint   value,    // border color
   )
```

Parameters

value

[in] Border color.

ColorText (Get method)

Returns the text color.

```
uint  ColorText()
```

Return Value

Text color.

ColorText (Set method)

Sets the text color.

```
void  ColorText(
   const uint  value,   // text color
   )
```

Parameters

value

[in] Text color.

ColorGrid (Get method)

Returns the grid color.

```
uint  ColorGrid()
```

Return Value

Grid color.

ColorGrid (Set method)

Sets the grid color.

```
void  ColorGrid(
   const uint  value,   // grid color
   )
```

Parameters

value

[in] Grid color.

MaxData (Get method)

Returns the maximum amount of data (series) allowed.

```
uint  MaxData()
```

Return Value

The maximum amount of data (series).

MaxData (Set method)

Sets the maximum amount of data (series) allowed.

```
void  MaxData(
   const uint   value,   // amount of data
   )
```

Parameters

value

[in] The maximum amount of data (series).

MaxDescrLen (Get method)

Returns the maximum length of the descriptors.

```
uint  MaxDescrLen()
```

Return Value

The value of the maximum length of the descriptors.

MaxDescrLen (Set method)

Sets the maximum length of the descriptors.

```
void  MaxDescrLen(
   const uint  value,   // maximum length
   )
```

Parameters

value

[in] The value of the maximum length of the descriptors.

ShowFlags (Get method)

Returns the visibility flag of the chart elements.

```
bool  ShowFlags()
```

Return Value

Value of the visibility flag of the chart elements.

ShowFlags (Set method)

Sets the visibility flag of the chart elements.

```
void  ShowFlags(
   const uint  flags,   // flag
   )
```

Parameters

flags

[in] Value of the visibility flag of the chart elements.

IsShowLegend

Returns and sets the visibility flag of the legend on the chart.

```
bool  IsShowLegend()
```

Return Value

true if the legend is visible, otherwise — false.

IsShowScaleLeft

Returns the visibility flag of the scale of values on the left.

```
bool  IsShowScaleLeft()
```

Return Value

true if the scale of values is visible, otherwise — false.

IsShowScaleRight

Returns the visibility flag of the scale of values on the right.

```
bool  IsShowScaleRight()
```

Return Value

true if the scale of values is visible, otherwise — false.

IsShowScaleTop

Returns the visibility flag of the scale of values at the top.

```
bool  IsShowScaleTop()
```

Return Value

true if the scale of values is visible, otherwise — false.

IsShowScaleBottom

Returns the visibility flag of the scale of values at the bottom.

```
bool  IsShowScaleBottom()
```

Return Value

true if the scale of values is visible, otherwise — false.

IsShowGrid

Returns the visibility flag of the grid on the chart.

```
bool  IsShowGrid()
```

Return Value

true if the grid is visible, otherwise – false.

IsShowDescriptors

Returns the visibility flag of the descriptors on the chart.

```
bool  IsShowDescriptors()
```

Return Value

true if the descriptors are visible, otherwise — false.

IsShowPercent

Returns the visibility flag of the percentages on the chart.

```
bool  IsShowPercent()
```

Return Value

true if the percentages are visible, otherwise — false.

VScaleMin (Get method)

Returns the minimum on the vertical scale of values.

```
double  VScaleMin()
```

Return Value

The minimum value on the vertical scale.

VScaleMin (Set method)

Sets the minimum on the vertical scale of values.

```
void  VScaleMin(
   const double  value,     // value on the vertical scale
   )
```

Parameters

value

[in] The minimum value.

VScaleMax

Returns the maximum on the vertical scale of values.

```
double  VScaleMax()
```

Return Value

The maximum value on the vertical scale.

VScaleMax

Sets the maximum on the vertical scale of values.

```
void  VScaleMax(
   const double  value,   //   value on the vertical scale
   )
```

Parameters

value

[in] The maximum value.

NumGrid

Returns the number of vertical scale divisions when plotting the chart grid.

```
uint  NumGrid()
```

Return Value

The number of divisions.

NumGrid

Sets the number of vertical scale divisions when plotting the chart grid.

```
void  NumGrid(
   const uint   value,   // number of divisions
   )
```

Parameters

value

[in] The number of divisions.

DataOffset

Returns the data offset value.

```
int  DataOffset()
```

Return Value

Data offset.

DataOffset

Sets the data offset value.

```
void  DataOffset(
   const int  value,   // offset
   )
```

Parameters

value

[in] Data offset.

DataTotal

Returns the total number of data series on the chart.

```
uint  DataTotal()
```

Return Value

The number of series.

DrawDescriptors

Virtual method for drawing descriptors.

```
virtual void  DrawDescriptors()
```

DrawData

Virtual method for drawing data series at the specified index.

```
virtual void  DrawData(
   const uint   idx=0,   // index
   )
```

Parameters

idx=0

[in] Index of the series.

Create

Virtual method that creates a graphical resource.

```
virtual bool  Create(
   const string        name,       // resource name
   const int           width,      // width
   const int           height,     // height
   ENUM_COLOR_FORMAT   clrfmt,     // format
   )
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt

[in] Color processing method. See the ResourceCreate() function description to learn more about color processing methods.

Return Value

true if successful, otherwise — false.

AllowedShowFlags

Sets the set of allowed visibility flags for chart elements.

```
void  AllowedShowFlags(
   const uint  flags,  // flags
   )
```

Parameters

flags

[in] Allowed flags.

ShowLegend

Sets the visibility flag value for the legend (FLAG_SHOW_LEGEND).

```
void  ShowLegend(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true — the legend becomes visible.
- false — the legend becomes invisible.

ShowScaleLeft

Sets the visibility flag value for the left scale (FLAG_SHOW_SCALE_LEFT).

```
void  ShowScaleLeft(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true – the left scale becomes visible.
- false – the left scale becomes invisible.

ShowScaleRight

Sets the visibility flag value for the right scale (FLAG_SHOW_SCALE_RIGHT).

```
void  ShowScaleRight(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true — the right scale becomes visible.
- false — the right scale becomes invisible.

ShowScaleTop

Sets the visibility flag value for the top scale (FLAG_SHOW_SCALE_TOP).

```
void  ShowScaleTop(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true — the top scale becomes visible.
- false — the top scale becomes invisible.

ShowScaleBottom

Sets the visibility flag value for the bottom scale (FLAG_SHOW_SCALE_BOTTOM).

```
void  ShowScaleBottom(
   const bool   flag,      // flag value
   )
```

Parameters

flag

[in] Flag value:

- true – the bottom scale becomes visible.
- false – the bottom scale becomes invisible.

ShowGrid

Sets the visibility flag value for the grid (FLAG_SHOW_GRID).

```
void  ShowGrid(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true — the grid becomes visible.
- false — the grid becomes invisible.

ShowDescriptors

Sets the visibility flag value for the descriptors (FLAG_SHOW_DESCRIPTORS).

```
void  ShowDescriptors(
   const bool   flag,     // flag value
   )
```

Parameters

flag

[in] Flag value:

- true – the descriptor becomes visible.
- false – the descriptor becomes invisible.

ShowValue

Sets the visibility flag for the values (FLAG_SHOW_VALUE).

```
void  ShowValue(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true – the value becomes visible.
- false – the value becomes invisible.

ShowPercent

Sets the visibility flag value for the percentages (FLAG_SHOW_PERCENT).

```
void  ShowPercent(
   const bool   flag,   // flag value
   )
```

Parameters

flag

[in] Flag value:

- true – the percentage becomes visible.
- false – the percentage becomes invisible.

LegendAlignment

Sets the text alignment for the legend.

```
void  LegendAlignment(
   const ENUM_ALIGNMENT  value,   // flag
   )
```

Parameters

value

[in] Takes one of the values of the ENUM_ALIGNMENT enumeration:

- ALIGNMENT_LEFT — alignment to the left.
- ALIGNMENT_TOP — alignment to the top.
- ALIGNMENT_RIGHT — alignment to the right.
- ALIGNMENT_BOTTOM — alignment to the bottom.

Accumulative

Sets the value accumulation flag for the series.

```
void  Accumulative(
   const bool   flag=true,   // flag value
   )
```

Parameters

flag=true

[in] Flag value:

- true – the current value of the series is replaced by the sum of all previous values.
- false – the standard mode for drawing series.

VScaleParams

Sets the parameters for the vertical scale of values.

```
void  VScaleParams(
   const double  max,    // maximum
   const double  min,    // minimum
   const uint    grid,   // number of divisions
   )
```

Parameters

max

[in] The minimum value.

min

[in] The maximum value.

grid

[in] The number of scale divisions.

DescriptorUpdate

Updates the value of the series descriptor (at the specified position).

```
bool  DescriptorUpdate(
   const uint     pos,     // index
   const string  descr,   // value
   )
```

Parameters

pos

[in] Index of the series — the serial number of its addition, starting with 0.

descr

[in] Descriptor value.

Return Value

true if successful, otherwise — false.

ColorUpdate

Updates the series colors (at the specified position).

```
bool  ColorUpdate(
   const uint  pos,   // index
   const uint  clr,   // color
   )
```

Parameters

pos

[in] Index of the series – the serial number of its addition, starting with 0.

clr

[in] Color value.

Return Value

true if successful, otherwise – false.

ValuesCheck

Auxiliary virtual method, performs internal calculations for plotting the chart.

```
virtual void  ValuesCheck()
```

Redraw

Virtual method for redrawing the chart.

```
virtual void  Redraw()
```

DrawBackground

Virtual method for redrawing the background.

```
virtual void  DrawBackground()
```

DrawLegend

Virtual method for redrawing the legend.

```
virtual void  DrawLegend()
```

DrawLegendVertical

Draws a vertical legend.

```
int  DrawLegendVertical(
   const int  w,   // width
   const int  h,   // height
   )
```

Parameters

w

[in] The maximum width of the text in the legend.

h

[in] The maximum height of the text in the legend.

Return Value

Width of the legend in pixels.

DrawLegendHorizontal

Draws a horizontal legend.

```
int  DrawLegendHorizontal(
   const int  w,   //
   const int  h,   //
   )
```

Parameters

w

[in] The maximum width of the text in the legend.

h

[in] The maximum height of the text in the legend.

Return Value

Height of the legend in pixels.

CalcScales

Virtual method for calculating the coordinates of labels for the scale of values.

```
virtual void  CalcScales()
```

DrawScales

Virtual method for redrawing the all scales of values.

```
virtual void  DrawScales()
```

DrawScaleLeft

Virtual method for redrawing the left scale of values.

```
virtual int  DrawScaleLeft(
   const bool  draw,   // flag
   )
```

Parameters

draw

[in] Flag that indicates if the scale needs to be redrawn.

Return Value

Width of the scale of values.

DrawScaleRight

Virtual method for redrawing the right scale of values.

```
virtual int  DrawScaleRight(
   const bool  draw,   // flag
   )
```

Parameters

draw

[in] Flag that indicates if the scale needs to be redrawn.

Return Value

Width of the scale of values.

DrawScaleTop

Virtual method for redrawing the top scale of values.

```
virtual int  DrawScaleTop(
   const bool  draw,   // flag
   )
```

Parameters

draw

[in] Flag that indicates if the scale needs to be redrawn.

Return Value

Height of the scale of values.

DrawScaleBottom

Virtual method for redrawing the bottom scale of values.

```
virtual int  DrawScaleBottom(
   const bool  draw,   // flag
   )
```

Parameters

draw

[in] Flag that indicates if the scale needs to be redrawn.

Return Value

Height of the scale of values.

DrawGrid

Virtual method for redrawing the grid.

```
virtual void  DrawGrid()
```

DrawChart

Virtual method for redrawing the chart.

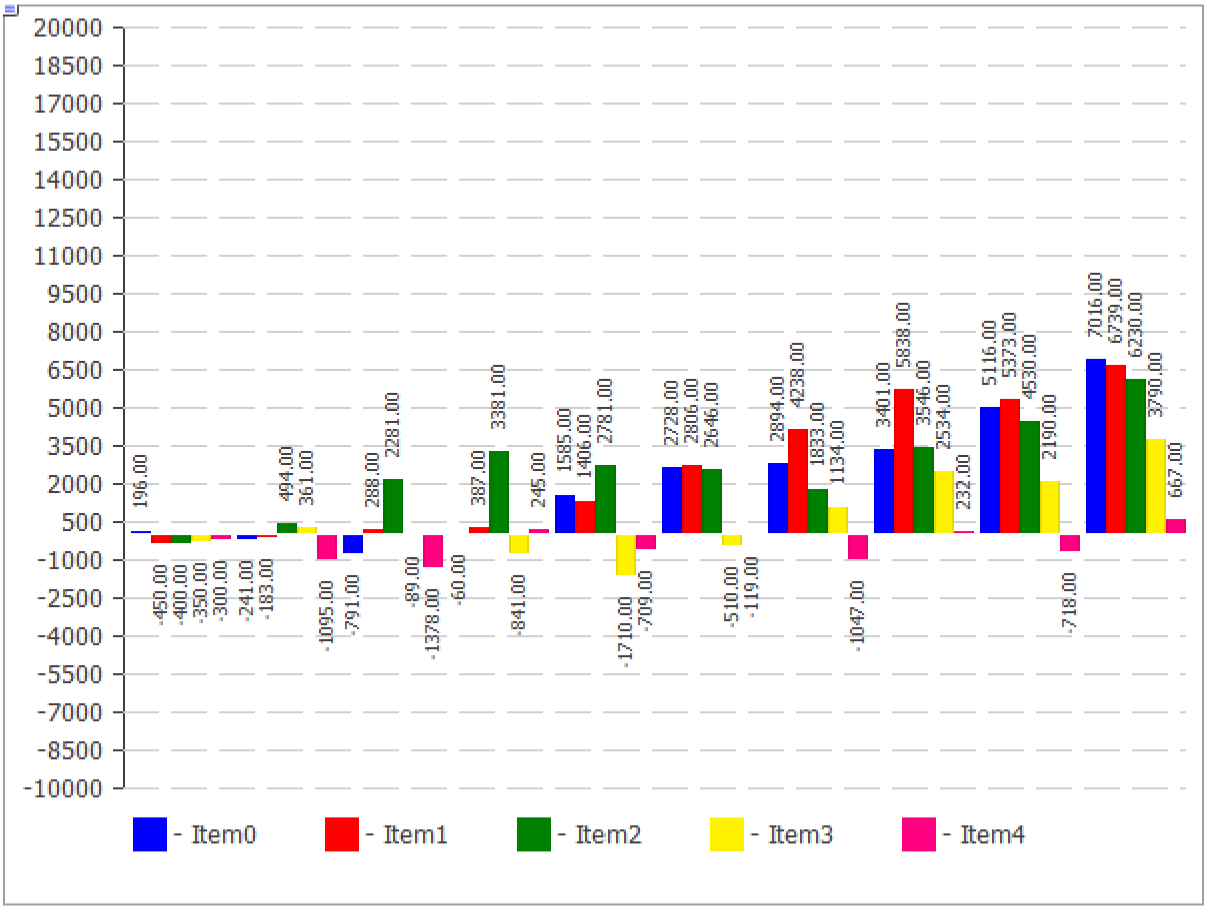
```
virtual void  DrawChart()
```

CHistogramChart

Class for plotting histograms.

Description

All methods for working with the plotting of histograms are implemented in this class. They can be used to set the column width and for configuring the work with data series. The methods for working with gradient filling of histogram columns are included, which allow to visualize the data more clearly.



The code of the above figure is provided below.

Declaration

```
class CHistogramChart : public CChartCanvas
```

Title

```
#include <Canvas\Charts\HistogramChart.mqh>
```

Inheritance hierarchy

CCanvas
    CChartCanvas
        CHistogramChart

Class methods

Method	Action
Gradient	Sets the flag indicating whether the gradient fill of the histogram columns will be applied.
BarGap	Set the value of the histogram offset from the origin.
BarMinSize	Sets the minimum width of the histogram columns.
BarBorder	Sets the flag indicating the need to draw the border for each column.
Create	Virtual method that creates a graphical resource.
SeriesAdd	Adds a new data series.
SeriesInsert	Inserts data series to the chart.
SeriesUpdate	Updates data series on the chart.
SeriesDelete	Deletes data series from the chart.
ValueUpdate	Updates the element value in the specified series.
DrawData	Virtual method that plots a histogram for the specified series.
DrawBar	Draws a histogram column as a filled rectangle.
GradientBrush	Creates a brush for the gradient fill.

Methods inherited from class CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

Methods inherited from class CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags,

Methods inherited from class CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

Example

```
//+------------------------------------------------------------------+
//|                                       HistogramChartSample.mq5 |
//|                   Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Example of using histogram"
//---
#include <Canvas\Charts\HistogramChart.mqh>
//+------------------------------------------------------------------+
//| inputs                                                           |
//+------------------------------------------------------------------+
input bool Accumulative=true;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   int k=100;
   double arr[10];
//--- create chart
   CHistogramChart chart;
   if(!chart.CreateBitmapLabel("SampleHistogramChart",10,10,600,450))
     {
      Print("Error creating histogram chart: ",GetLastError());
      return(-1);
     }
   if(Accumulative)
     {
      chart.Accumulative();
      chart.VScaleParams(20*k*10,-10*k*10,20);
     }
   else
      chart.VScaleParams(20*k,-10*k,20);
   chart.ShowValue(true);
   chart.ShowScaleTop(false);
   chart.ShowScaleBottom(false);
   chart.ShowScaleRight(false);
   chart.ShowLegend();
   for(int j=0;j<5;j++)
     {
      for(int i=0;i<10;i++)
        {
         k=-k;
         if(k>0)
            arr[i]=k*(i+10-j);
         else
            arr[i]=k*(i+10-j)/2;
        }
      chart.SeriesAdd(arr,"Item"+IntegerToString(j));
     }
//--- play with values
   while(!IsStopped())
     {
      int i=rand()%5;
      int j=rand()%10;
      k=rand()%3000-1000;
      chart.ValueUpdate(i,j,k);
      Sleep(200);
     }
//--- finish
   chart.Destroy();
   return(0);
  }
```

Gradient

Sets the flag indicating whether the gradient fill of the histogram columns will be applied.

```
void  Gradient(
   const bool   flag=true,   // flag value
   )
```

Parameters

flag=true

Flag value: true if the gradient fill is enabled, otherwise – false.

BarGap

Set the value of the histogram offset from the origin.

```
void  BarGap(
   const uint   value,   // offset
   )
```

Parameters

value

[in] Value of the histogram offset.

BarMinSize

Sets the minimum width of the histogram columns.

```
void  BarMinSize(
   const uint   value,   // minimum width
   )
```

Parameters

value

[in] The minimum width.

BarBorder

Sets the flag indicating the need to draw the border for each column.

```
void  BarBorder(
   const uint   value,   // flag
   )
```

Parameters

value

[in] Flag value:

- true — borders will be drawn
- false — borders will not be drawn

Create

Virtual method that creates a graphical resource.

```
virtual bool  Create(
   const string        name,     // name
   const int           width,    // width
   const int           height,   // height
   ENUM_COLOR_FORMAT   clrfmt,   // format
   )
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt

[in] Color processing method. See the ResourceCreate() function description to learn more about color processing methods.

Return Value

true if successful, otherwise – false.

SeriesAdd

Adds a new data series.

```
bool  SeriesAdd(
   const double&  value[],   // values
   const string  descr,      // label
   const uint    clr,        // color
   )
```

Parameters

value[]

[in] Data series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise – false.

SeriesInsert

Inserts data series to the chart.

```
bool  SeriesInsert(
   const uint     pos,         // index
   const double&  value[],   // values
   const string  descr,      // label
   const uint     clr,         // color
   )
```

Parameters

pos

[in] Index for insertion.

value[]

[in] Dara series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise — false.

SeriesUpdate

Updates data series on the chart.

```
bool  SeriesUpdate(
   const uint    pos,         // index
   const double  &value[],    // values
   const string  descr,       // label
   const uint    clr,         // color
   )
```

Parameters

pos

[in] Index of the series – the serial number of its addition, starting with 0.

&value[]

[in] New values for the data series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise false.

SeriesDelete

Deletes data series from the chart.

```
bool  SeriesDelete(
   const uint  pos,   // index
   )
```

Parameters

pos

[in] Index of the series – the serial number of its addition, starting with 0.

Return Value

true if successful, otherwise – false.

ValueUpdate

Updates the specified value in the specified series.

```
bool  ValueUpdate(
   const uint  series,   // index of the series
   const uint  pos,      // index of the element
   double      value,    // value
   )
```

Parameters

series

[in] Index of the series – the serial number of its addition, starting with 0.

pos

[in] Index of element in the series.

value

[in] New value.

Return Value

true if successful, otherwise – false.

DrawData

Virtual method that plots a histogram for the specified series.

```
virtual void  DrawData(
   const uint  index,   // index
   )
```

Parameters

index

[in] Index of the series – the serial number of its addition, starting with 0.

DrawBar

Draws a histogram column as a filled rectangle.

```
void  DrawBar(
   const int    x,     // X coordinate
   const int    y,     // Y coordinate
   const int    w,     // width
   const int    h,     // height
   const uint   clr,   // color
   )
```

Parameters

x

[in] X coordinate of the top left point of the rectangle.

y

[in] Y coordinate of the top left point of the rectangle.

w

[in] The width of the rectangle.

h

[in] The height of the rectangle.

clr

[in] The color of the rectangle.

GradientBrush

Creates a brush for the gradient fill.

```
void  GradientBrush(
   const int    size,       // size
   const uint   fill_clr,   // fill color
   )
```

Parameters

size

[in] Brush thickness

fill_clr

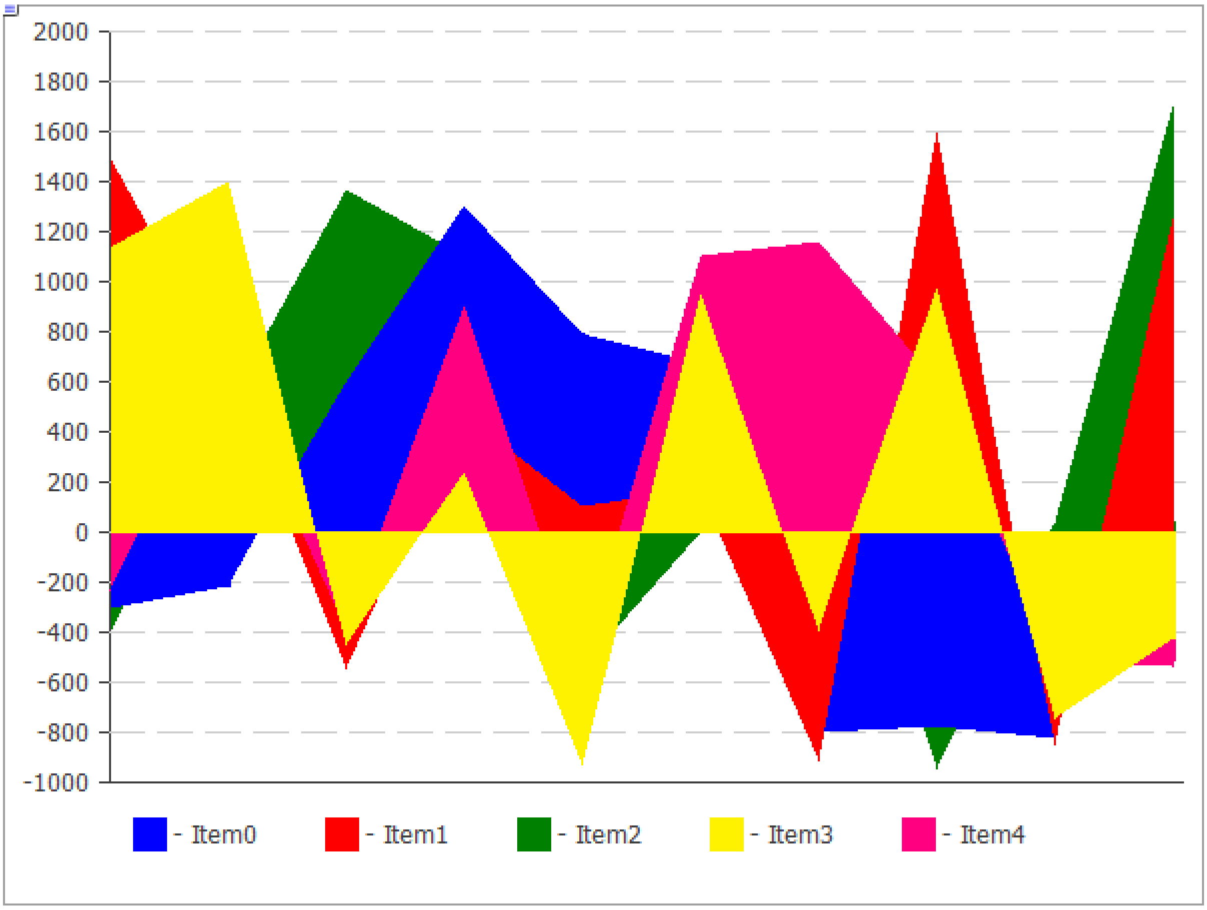
[in] Fill color

CLineChart

A class for plotting curves.

Description

The methods included in this class are designed for working with curves on the chart. It features the ability to fill the area limited by the plotted curve.



The code of the above figure is provided below.

Declaration

```
   class CLineChart : public CChartCanvas
```

Title

```
   #include <Canvas\Charts\LineChart.mqh>
```

Inheritance hierarchy

- CCanvas
  - CChartCanvas
    - CLineChart

Class methods

Method	Action
Filled	Sets the flag for filling the area under the curve defined by the data series.
Create	Creates a graphical resource.
SeriesAdd	Adds a new data series.
SeriesInsert	Inserts data series to the chart.
SeriesUpdate	Updates data series on the chart.
SeriesDelete	Deletes data series from the chart.
ValueUpdate	Updates the specified value in the specified series.
DrawChart	Virtual method which draws a curve and all its elements.
DrawData	Virtual method which draws a curve for the specified series.
CalcArea	Calculates the area under the curve defined by the data series.

Methods inherited from class CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

Methods inherited from class CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags, ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

Example

```
//+------------------------------------------------------------------+
//|                                              LineChartSample.mq5 |
//|                        Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Example of using line chart"
//---
#include <Canvas\Charts\LineChart.mqh>
//+------------------------------------------------------------------+
//| inputs                                                           |
//+------------------------------------------------------------------+
input bool Accumulative=false;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart(void)
  {
   int k=100;
   double arr[10];
//--- create chart
   CLineChart chart;
//--- create chart
   if(!chart.CreateBitmapLabel("SampleHistogrammChart",10,10,600,450))
     {
      Print("Error creating line chart: ",GetLastError());
      return(-1);
     }
   if(Accumulative)
     {
      chart.Accumulative();
      chart.VScaleParams(20*k*10,-10*k*10,20);
     }
   else
      chart.VScaleParams(20*k,-10*k,15);
   chart.ShowScaleTop(false);
   chart.ShowScaleRight(false);
   chart.ShowLegend();
   chart.Filled();
   for(int j=0;j<5;j++)
     {
      for(int i=0;i<10;i++)
        {
         k=-k;
         if(k>0)
            arr[i]=k*(i+10-j);
         else
            arr[i]=k*(i+10-j)/2;
        }
      chart.SeriesAdd(arr,"Item"+IntegerToString(j));
     }
//--- play with values
   while(!IsStopped())
     {
      int i=rand()%5;
      int j=rand()%10;
      k=rand()%3000-1000;
      chart.ValueUpdate(i,j,k);
      Sleep(200);
     }
//--- finish
   chart.Destroy();
   return(0);
  }
```

Filled

Sets the flag indicating whether it is necessary to fill the area under the curve defined by the data series.

```
void  Filled(
   const bool   flag=true,   // flag
   )
```

Parameters

flag=true

[in] Flag value:

- true — fill the area under the curve
- false — do not fill the area under the curve

Create

Virtual method that creates a graphical resource.

```
virtual bool  Create(
   const string        name,     // name
   const int           width,    // width
   const int           height,   // height
   ENUM_COLOR_FORMAT   clrfmt,   // format
   )
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt

[in] Color processing method. See the ResourceCreate() function description to learn more about color processing methods.

Return Value

true if successful, otherwise – false.

SeriesAdd

Adds a new data series.

```
bool  SeriesAdd(
   const double&  value[],   // values
   const string  descr,      // label
   const uint    clr,        // color
   )
```

Parameters

value[]

[in] Dara series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise – false.

SeriesInsert

Inserts data series to the chart.

```
bool  SeriesInsert(
   const uint      pos,         // index
   const double&   value[],     // values
   const string    descr,       // label
   const uint      clr,         // color
   )
```

Parameters

pos

[in] Index for insertion.

value[]

[in] Dara series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise – false.

SeriesUpdate

Updates data series on the chart.

```
bool  SeriesUpdate(
   const uint      pos,         // index
   const double&   value[],     // values
   const string    descr,       // label
   const uint      clr,         // color
   )
```

Parameters

pos

[in] Index of the series — the serial number of its addition, starting with 0.

value[]

[in] New values for the data series.

descr

[in] Series label.

clr

[in] Series display color.

Return Value

true if successful, otherwise — false.

SeriesDelete

Deletes data series from the chart.

```
bool  SeriesDelete(
   const uint  pos,   // index
   )
```

Parameters

pos

[in] Index of the series – the serial number of its addition, starting with 0.

Return Value

true if successful, otherwise – false.

ValueUpdate

Updates the specified value in the specified series.

```
bool  ValueUpdate(
   const uint  series,   // index of the series
   const uint  pos,      // index of the element
   double      value,    // value
   )
```

Parameters

series

[in] Index of the series – the serial number of its addition, starting with 0.

pos

[in] Index of element in the series.

value

[in] New value.

Return Value

true if successful, otherwise – false.

DrawChart

Virtual method which draws a curve and all its elements.

```
virtual void  DrawChart()
```

DrawData

Virtual method which draws a curve for the specified series.

```
virtual void  DrawData(
   const uint  index,   // index
   )
```

Parameters

index

[in] Index of the series – the serial number of its addition, starting with 0.

CalcArea

Calculates the area under the curve defined by the data series.

```
double  CalcArea(
   const uint  index,   // index
   )
```

Parameters

index

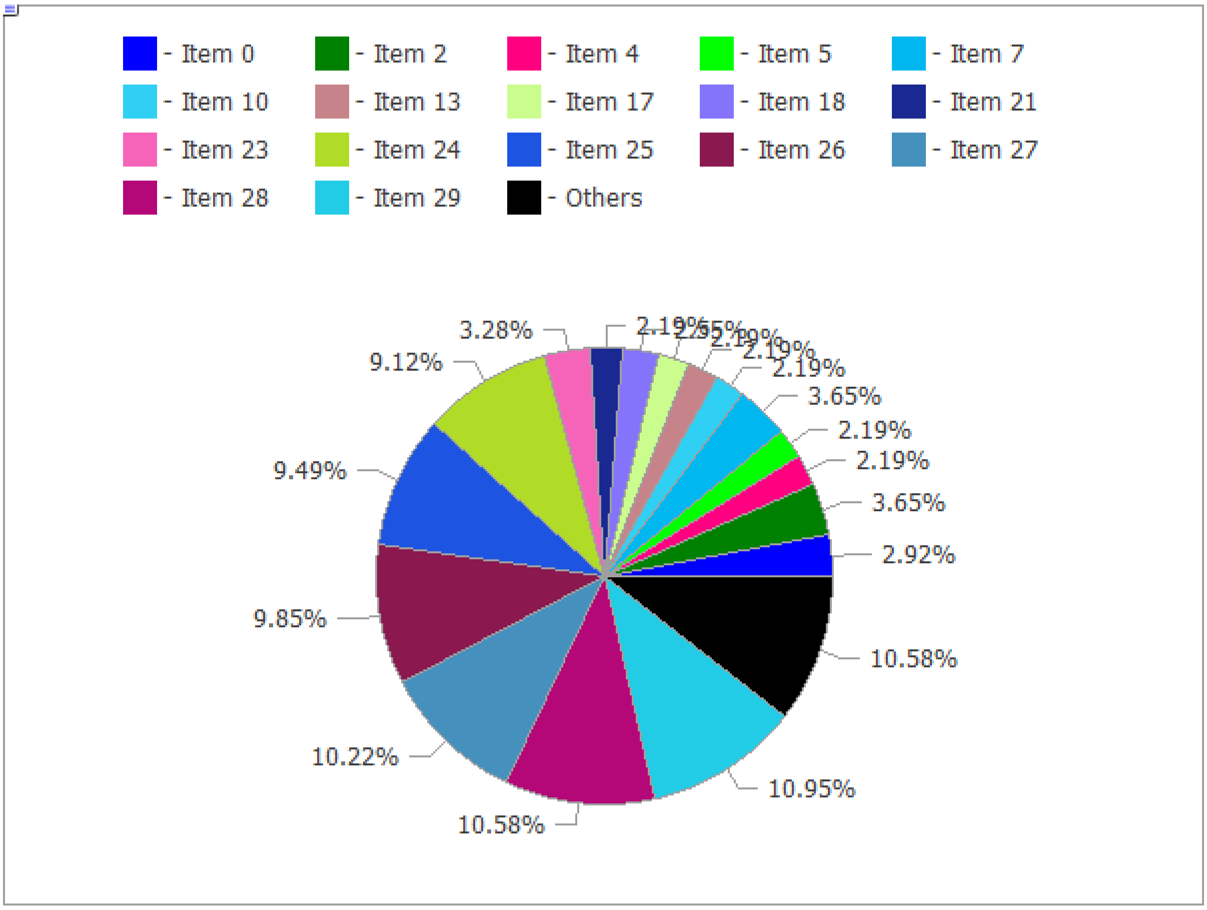
[in] Index of the series – the serial number of its addition, starting with 0.

Return Value

Area of the figure limited by the curve which is defined by the data series.

CPieChart

Class for plotting pie charts.



The code of the above figure is provided below.

Description

The methods included in this class are designed for full-scale operation with pie charts, from the creating a graphical resource to designing labels to segments.

Declaration

```
class CPieChart : public CChartCanvas
```

Title

```
#include <Canvas\Charts\PieChart.mqh>
```

Inheritance hierarchy

CCanvas

  CChartCanvas

    CPieChart

Class methods

Method	Action
Create	Virtual method that creates a graphical resource.
SeriesSet	Sets a series of values that will be shows on the pie chart.
ValueAdd	Adds a new value to the pie chart (to the end).
ValueInsert	Inserts a new value to the pie chart (at the specified position).
ValueUpdate	Updates the value on the pie chart (at the specified position).
ValueDelete	Removes a value from the pie chart (at the specified position).
DrawChart	Virtual method which draws a pie chart and all its elements.
DrawPie	Draws a segment of the pie chart, which corresponds to a specified value.
LabelMake	Generates a segment label based on its value and the original label.

Methods inherited from class CCanvas

CreateBitmap, CreateBitmap, CreateBitmapLabel, CreateBitmapLabel, Attach, Attach, Destroy, ChartObjectName, ResourceName, Width, Height, Update, Resize, Erase, PixelGet, PixelSet, LineVertical, LineHorizontal, Line, Polyline, Polygon, Rectangle, Triangle, Circle, Ellipse, Arc, Arc, Arc, Pie, Pie, FillRectangle, FillTriangle, FillPolygon, FillCircle, FillEllipse, Fill, Fill, PixelSetAA, LineAA, PolylineAA, PolygonAA, TriangleAA, CircleAA, EllipseAA, LineWu, PolylineWu, PolygonWu, TriangleWu, CircleWu, EllipseWu, LineThickVertical, LineThickHorizontal, LineThick, PolylineThick, PolygonThick, PolylineSmooth, PolygonSmooth, FontSet, FontNameSet, FontSizeSet, FontFlagsSet, FontAngleSet, FontGet, FontNameGet, FontSizeGet, FontFlagsGet, FontAngleGet, TextOut, TextWidth, TextHeight, TextSize, GetDefaultColor, TransparentLevelSet, LoadFromFile, LineStyleGet, LineStyleSet

Methods inherited from class CChartCanvas

ColorBackground, ColorBackground, ColorBorder, ColorBorder, ColorText, ColorText, ColorGrid, ColorGrid, MaxData, MaxData, MaxDescrLen, MaxDescrLen, AllowedShowFlags, ShowFlags, ShowFlags, IsShowLegend, IsShowScaleLeft, IsShowScaleRight, IsShowScaleTop, IsShowScaleBottom, IsShowGrid, IsShowDescriptors, IsShowPercent, ShowLegend, ShowScaleLeft, ShowScaleRight, ShowScaleTop, ShowScaleBottom, ShowGrid, ShowDescriptors, ShowValue, ShowPercent, LegendAlignment, Accumulative, VScaleMin, VScaleMin, VScaleMax, VScaleMax, NumGrid, NumGrid, VScaleParams, DataOffset, DataOffset, DataTotal, DescriptorUpdate, ColorUpdate

Example

```
//+------------------------------------------------------------------+
//|                                               PieChartSample.mq5 |
//|                   Copyright 2009-2017, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright   "2009-2017, MetaQuotes Software Corp."
#property link        "http://www.mql5.com"
#property description "Example of using pie chart"
//---
#include <Canvas\Charts\PieChart.mqh>
//+------------------------------------------------------------------+
//| inputs                                                           |
//+------------------------------------------------------------------+
input int      Width=600;
input int      Height=450;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int OnStart(void)
  {
//--- check
   if(Width<=0 || Height<=0)
     {
      Print("Too simple.");
      return(-1);
     }
//--- create chart
   CPieChart pie_chart;
   if(!pie_chart.CreateBitmapLabel("PieChart",10,10,Width,Height))
     {
      Print("Error creating pie chart: ",GetLastError());
      return(-1);
     }
   pie_chart.ShowPercent();
//--- draw
   for(uint i=0;i<30;i++)
     {
      pie_chart.ValueAdd(100*(i+1),"Item "+IntegerToString(i));
      Sleep(10);
     }
   Sleep(2000);
//--- disable legend
   pie_chart.LegendAlignment(ALIGNMENT_LEFT);
   Sleep(2000);
//--- disable legend
   pie_chart.LegendAlignment(ALIGNMENT_RIGHT);
   Sleep(2000);
//--- disable legend
   pie_chart.LegendAlignment(ALIGNMENT_TOP);
   Sleep(2000);
//--- disable legend
   pie_chart.ShowLegend(false);
   Sleep(2000);
//--- disable percentage
   pie_chart.ShowPercent(false);
   Sleep(2000);
//--- disable descriptors
   pie_chart.ShowDescriptors(false);
   Sleep(2000);
//--- enable all
   pie_chart.ShowLegend();
   pie_chart.ShowValue();
   pie_chart.ShowDescriptors();
   Sleep(2000);
//--- or like this
   pie_chart.ShowFlags(FLAG_SHOW_LEGEND|FLAG_SHOW_DESCRIPTORS|FLAG_SHOW_PERCENT);
   uint total=pie_chart.DataTotal();
//--- play with values
   for(uint i=0;i<total && !IsStopped();i++)
     {
      pie_chart.ValueUpdate(i,100*(rand()%10+1));
      Sleep(1000);
     }
//--- play with colors
   for(uint i=0;i<total && !IsStopped();i++)
     {
      pie_chart.ColorUpdate(i%total,RandomRGB());
      Sleep(1000);
     }
//--- rotate
   while(!IsStopped())
     {
      pie_chart.DataOffset(pie_chart.DataOffset()+1);
      Sleep(200);
     }
//--- finish
   pie_chart.Destroy();
   return(0);
  }
//+------------------------------------------------------------------+
//| Random RGB color                                                 |
//+------------------------------------------------------------------+
uint RandomRGB(void)
  {
   return(XRGB(rand()%255,rand()%255,rand()%255));
  }
```

Create

Virtual method that creates a graphical resource.

```
virtual bool  Create(
   const string        name,     // name
   const int           width,    // width
   const int           height,   // height
   ENUM_COLOR_FORMAT   clrfmt,   // format
   )
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt

[in] Color processing method. See the ResourceCreate() function description to learn more about color processing methods.

Return Value

true if successful, otherwise – false.

SeriesSet

Sets a series of values that will be shows on the pie chart.

```
bool  SeriesSet(
   const double&  value[],   // values
   const string&  text[],    // labels
   const uint&    clr[],     // color
   )
```

Parameters

value[]

[in] Array of values.

text[]

[in] Array of value labels.

clr[]

[in] Array of value colors.

Return Value

true if successful, otherwise – false.

ValueAdd

Adds a new value to the pie chart (to the end).

```
bool  ValueAdd(
   const double  value,   // value
   const string  descr,   // label
   const uint    clr,     // color
   )
```

Parameters

value

[in] Value.

descr

[in] Value label.

clr

[in] Value color.

Return Value

true if successful, otherwise — false.

ValueInsert

Inserts a new value to the pie chart (at the specified position).

```
bool  ValueInsert(
   const uint    pos,     // index
   const double  value,   // value
   const string  descr,   // label
   const uint    clr,     // color
   )
```

Parameters

pos

[in] Index for insertion.

value

[in] Value.

descr

[in] Value label.

clr

[in] Value color.

Return Value

true if successful, otherwise – false.

ValueUpdate

Updates the value on the pie chart (at the specified position).

```
bool  ValueUpdate(
   const uint    pos,     // index
   const double  value,   // value
   const string  descr,   // label
   const uint    clr,     // color
   )
```

Parameters

pos

[in] Index of the value – the serial number of its addition, starting with 0.

value

[in] Value.

descr

[in] Value label.

clr

[in] Value color.

Return Value

true if successful, otherwise – false.

ValueDelete

Removes a value from the pie chart (at the specified position).

```
bool  ValueDelete(
   const uint   pos,    // index
   )
```

Parameters

pos

[in] Index of the value – the serial number of its addition, starting with 0.

Return Value

true if successful, otherwise – false.

DrawChart

Virtual method which draws a pie chart and all its elements.

```
virtual void  DrawChart()
```

DrawPie

Draws a segment of the pie chart, which corresponds to a specified value.

```
void  DrawPie(
   double       fi3,   // angle of ray from pie chart center, which defines the first
   double       fi4,   // angle of ray from pie chart center, which defines the second
   int          idx,   // index
   CPoint&      p[],  //
   const uint   clr,   //
   )
```

Parameters

fi3

[in] Angle in radians, which defines the first boundary of the arc.

fi4

[in] Angle in radians, which defines the second boundary of the arc.

idx

[in] Index of the value the segment corresponds to.

p[]

[in] Array of reference points (x, y) for plotting the segments.

clr

[in] Color of the segment.

LabelMake

Generates a segment label based on its value and the original label.

```
string  LabelMake(
   const string  text,      // label
   const double  value,     // value
   const bool    to_left,   // flag
   )
```

Parameters

text

[in] Label.

value

[in] Value.

to_left

[in] Defines the order of the label layout:

- true – label, then value.
- false – value, then label.

Return Value

Label of the segment.

3D graphics

The section features the classes for the three-dimensional graphics development. The classes are based on the functions for working with DirectX. CCnavas3D is a base class containing the methods for managing camera and lighting, as well as featuring the manager of graphic resources – textures, shaders, vertex buffers, indexes and shader parameters.

To start working with the library, simply read the article How to create 3D graphics using DirectX in MetaTrader 5.

Besides, there are the classes of the scene base objects, such as a box, a three-dimensional surface on user data and an arbitrary grid.

CCanvas3D

CCanvas3D is a class for simplified creation and visualization of 3D objects on a chart.

Description

CCanvas3D greatly simplifies creation and visualization of large amounts of data in the form of animated 3D graphics. The class contains the methods for managing camera and lighting, as well as features the resource manager for creating graphic resources: textures, shaders, vertex buffers, indexes, and shader parameters.

Besides, the library contains the classes of the scene base objects, such as a box, a three-dimensional surface on user data, or an arbitrary grid.

A video card should support DX 11 and Shader Model 5.0 for the functions to work.

Declaration

```
   class CCanvas
```

Title

```
   #include <Canvas\Canvas.mqh>
```

Inheritance hierarchy

CCanvas

   CCanvas3D

Class methods by groups

Creating/deleting	Description
Create	Creates a graphic resource for rendering a 3D scene without binding to a chart object.
Attach	Gets a graphical resource from the OBJ_BITMAP_LABEL object and attaches it to an instance of the CCanvas class.
ObjectAdd	Adds an object to a 3D scene for subsequent rendering.
Destroy	Destroys a graphic resource and releases a graphic 3D context.
Light	
AmbientColorSet	Sets the color and intensity of the ambient all-round lighting.
AmbientColorGet	Gets the color and intensity of the ambient all-round lighting.
LightDirectionSet	Sets the direction of a directed light source.
LightDirectionGet	Gets the direction of a directed light source.
LightColorSet	Sets the color and intensity of a directed light source.

Creating/deleting	Description
LightColorGet	Gets the color and intensity of a directed light source.
Camera	
ProjectionMatrixSet	Calculates and sets a 3D coordinate projection matrix to a 2D frame.
ProjectionMatrixGet	Gets a 3D scene projection matrix to a 2D frame.
ViewMatrixSet	Sets a 3D scene view matrix.
ViewMatrixGet	Returns a 3D scene view matrix.
ViewPositionSet	Sets a viewpoint on a 3D scene.
ViewRotationSet	Sets the direction of a gaze at a 3D scene.
ViewTargetSet	Sets the coordinates of the point a gaze is directed at.
ViewUpDirectionSet	Sets the direction of the upper frame border in 3D space.
Rendering	
Render	Renders all scene objects in the frame inner buffer for subsequent display.
RenderBegin	Prepares a graphic context for rendering a new frame.
RenderEnd	Copies a rendered frame to the inner buffer and updates a chart image if necessary.
Getting resources	
DXContext	Gets the graphic context handle.
DXDispatcher	Gets the resource dispatcher handle.
InputScene	Gets the pointer to the buffer of scene parameters.

AmbientColorGet

Gets the color and intensity of the ambient all-round lighting.

```
void  AmbientColorGet(
   DXColor  &ambient_color      // all-round lighting color and intensity
   );
```

Parameters

&ambient_color

[out] All-round lighting color.

Return Value

No

Note

Intensity is stored in the alpha channel of the DXColor structure.

AmbientColorSet

Sets the color and intensity of the ambient all-round lighting.

```
void  AmbientColorSet(
   const DXColor  &ambient_color      // all-round lighting color and intensity
   );
```

Parameters

&ambient_color

[in] All-round lighting color.

Return Value

No

Note

Intensity is set in the alpha channel of the DXColor structure.

Attach

Gets the graphical resource from an OBJ_BITMAP_LABEL object and attaches it to an instance of the CCanvas class.

```
bool  Attach(
   const long          chart_id,                              // chart ID
   const string        objname,                               // object name
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // color handling method
   )
```

Creates a graphical resource for an OBJ_BITMAP_LABEL object and attaches it to an instance of the CCanvas class.

```
bool  Attach(
   const long          chart_id,                              // chart ID
   const string        objname,                               // object name
   const int           width,                                 // image width in pixels
   const int           height,                                // image height in pixels
   ENUM_COLOR_FORMAT   clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // color handling method
   )
```

Parameters

chart_id

[in] Chart ID.

objname

[in] Name of the graphical object.

width

[in] Frame width in a resource.

height

[in] Frame height.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color handling method. See ResourceCreate() function description to learn more about color handling methods.

Note

true - if successful, false - if failed to add a graphic object.

Create

Creates a graphic resource for rendering a 3D scene without binding to a chart object.

```
virtual bool  Create(
   const string       name,                                  // graphical object name
   const int          width,                                 // width
   const int          height,                                // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // color format
   );
```

Parameters

name

[in] Graphical object name.

width

[in] Frame width.

height

[in] Frame height.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color handling method. See ResourceCreate() function description to learn more about color handling methods.

Note

true - if a resource is created, otherwise - false.

Destroy

Destroys a graphic resource and releases a graphic 3D context.

```
virtual void  Destroy()
```

Return Value

No

Note

If a graphical resource has been bound to a chart object, the latter is deleted.

DXContext

Gets the graphic context handle.

```
int  DXContext()
```

Return Value

Graphic context handle.

DXDispatcher

Gets the resource dispatcher handle.

```
CDXDispatcher*  DXDispatcher()
```

Return Value

Resource dispatcher handle.

InputScene

Gets the pointer to the buffer of scene parameters.

```
CDXInput*   InputScene()
```

Return Value

Pointer to the buffer of scene parameters.

LightColorGet

Gets the color and intensity of a directed light source.

```
void  LightColorGet(
   DXColor  &light_color      // directional lighting color and intensity
   );
```

Parameters

&light_color

[out] Directional lighting color and intensity.

Return Value

None.

Note

Intensity is stored in the alpha channel of the DXColor structure.

LightColorSet

Sets the color and intensity of a directed light source.

```
void  LightColorSet(
   const DXColor  &light_color      // directional lighting color and intensity
   );
```

Parameters

&light_color

[in] Directional lighting color and intensity.

Return Value

None.

Note

Intensity is set in the alpha channel of the DXColor structure.

LightDirectionGet

Gets the direction of a directed light source.

```
void  LightDirectionGet(
   DXVector3  &light_direction      // direction vector
   );
```

Parameters

&light_direction

[out] Direction vector.

Return Value

None.

LightDirectionSet

Sets the direction of a directed light source.

```
void  LightDirectionSet(
   const DXVector3  &light_direction      // direction vector
   );
```

Parameters

&light_direction

[in] Direction vector.

Return Value

None.

ObjectAdd

Adds an object to a 3D scene for subsequent rendering.

```
bool  ObjectAdd(
   CDXObject   *object       // pointer to the object
   );
```

Parameters

**object*

[in] Pointer to an instance of the class derived from the CDXObject abstract class.

Return Value

true – if successful, false - if failed to add a 3D graphic object.

ProjectionMatrixGet

Gets a 3D scene projection matrix to a 2D frame.

```
void  ProjectionMatrixGet(
   DXMatrix  &projection_matrix       // projection matrix
   );
```

Parameters

&projection_matrix

[out] Projection matrix.

Return Value

None.

ProjectionMatrixSet

Calculates and sets a 3D coordinate projection matrix to a 2D frame.

```
void  ProjectionMatrixSet(
   float  fov,                // field of view
   float  aspect_ratio,       // frame aspect ratio
   float  z_near,             //
   float  z_far               //
   );
```

Parameters

fov

[in] Field of view width in radians to create a scene projection.

aspect_ratio

[in] 2D frame aspect ratio.

z_near

[in] Distance to the near clipping plane.

z_far

[in] Distance to the far clipping plane.

Return Value

None.

Note

2D frame displays only projections of 3D objects falling into the specified field of view and located between the near and far clipping planes.

Render

Renders all scene objects in the frame inner buffer for subsequent display.

```
bool  Render(
   uint   flags,                  // combination of flags
   uint   background_color=0      // background color
   );
```

Parameters

flags

[in] Combination of flags that sets the rendering mode. Possible values:
DX_CLEAR_COLOR – clear the image buffer using *background_color*.
DX_CLEAR_DEPTH – clear the depth buffer.

background_color=0

[in] 3D scene background color.

Return Value

true – if successful, false – if failed to render.

Note

Calling Render() does not update a scene on a chart. Instead, it only updates the inner buffer of the image. The Update() method should be explicitly called to render the updated frame.

Render() features the RenderBegin and RenderEnd() calls.

RenderBegin

Prepares a graphic context for rendering a new frame.

```
virtual bool  RenderBegin(
   uint   flags,                    // combination of flags
   uint   background_color=0        // background color
   );
```

Parameters

flags

[in] Combination of flags that sets the rendering mode. Possible values:
DX_CLEAR_COLOR – clear the image buffer using *background_color*.
DX_CLEAR_DEPTH – clear the depth buffer.

background_color=0

[in] 3D scene background color.

Return Value

true – if successful, false - if failed to update shader inputs.

RenderEnd

Copies a rendered frame to the inner buffer and updates a chart image if necessary.

```
virtual bool  RenderEnd(
   bool   redraw=false      // update flag
   );
```

Parameters

redraw=false

[in] Flag of a chart redrawing necessity.

Return Value

true - successful, otherwise - false.

ViewMatrixGet

Returns a 3D scene view matrix.

```
void  ViewMatrixGet(
   DXMatrix  &view_matrix      // view matrix
   );
```

Parameters

&view_matrix

[out] View matrix setting the camera position and direction in 3D space.

Return Value

None.

ViewMatrixSet

Sets a 3D scene view matrix.

```
void  ViewMatrixSet(
   const DXMatrix  &view_matrix      // view matrix
   );
```

Parameters

&view_matrix

[in] View matrix setting the camera position and direction in 3D space.

Return Value

None.

ViewPositionSet

Sets a viewpoint on a 3D scene.

```
void  ViewPositionSet(
   const DXVector3  &position        // viewpoint position
   );
```

Parameters

&position

[in] Setting a viewpoint position on a 3D scene.

Return Value

None.

Note

Setting a viewpoint position using ViewPositionSet() changes the view matrix obtained in ViewMatrixGet().

ViewRotationSet

Sets the direction of a gaze at a 3D scene.

```
void  ViewRotationSet(
   const DXVector3  &rotation      // vector of turning angles
   );
```

Parameters

&rotation

[in] Vector setting Euler angles to calculate the direction of a gaze at a 3D scene.

Return Value

None.

Note

Setting the gaze direction using ViewRotationSet() changes the view matrix obtained in ViewMatrixGet().

ViewTargetSet

Sets the coordinates of the point a gaze is directed at.

```
void  ViewTargetSet(
   const DXVector3  &target       // target coordinates
   );
```

Parameters

&target

[in] Coordinates of the point a gaze is directed at.

Return Value

None.

Note

Used to fix the gaze at one scene point when moving the viewpoint.

Setting a new target coordinate using ViewRotationSet() changes the view matrix obtained in ViewMatrixGet().

ViewTargetSet() is used together with ViewUpDirectionSet() to define the gaze direction.

ViewUpDirectionSet

Sets the direction of the upper frame border in 3D space.

```
void  ViewUpDirectionSet(
   const DXVector3  &up_direction       // top direction
   );
```

Parameters

&up_direction

[in] Direction of the upper part of the frame in 3D space.

Return Value

None.

Note

Setting a new direction using ViewUpDirectionSet() changes the view matrix obtained in ViewMatrixGet().

ViewUpDirectionSet() is used together with ViewTargetSet() to define the gaze direction.

CChart

CChart is a class for simplified access to "Chart" graphic object properties.

Description

CChart class provides access to "Chart" object properties.

Declaration

```
class CChart : public CObject
```

Title

```
#include <Charts\Chart.mqh>
```

Inheritance hierarchy

CObject

CChart

Class Methods by Groups

Access to protected data	
ChartID	Gets identifier of the chart
General properties	
Mode	Gets/sets the value of "Mode" property (bars, candles, or line)
Foreground	Gets/sets the value of "Foreground" property
Shift	Gets/sets the value of "Shift" property
ShiftSize	Gets/sets the value of "ShiftSize" property (in percents)
AutoScroll	Gets/sets the value of "AutoScroll" property
Scale	Gets/sets the value of "Scale" property
ScaleFix	Gets/sets the value of "ScaleFix" property (fixed chart scale or not)
ScaleFix_11	Gets/sets the value of "ScaleFix_11" property (chart scale is 1:1, or not)
FixedMax	Gets/sets the value of "FixedMax" property (fixed maximal price)
FixedMin	Gets/sets the value of "FixedMin" property (fixed minimal price)
ScalePPB	Gets/sets the value of "ScalePPB" property (scale is "point per bar" or not)

Access to protected data	
PointsPerBar	Gets/sets the value of "PointsPerBar" property (in points per bar)
Show properties	
ShowOHLC	Gets/sets the value of "ShowOHLC" property
ShowLineBid	Gets/sets the value of "ShowLineBid" property
ShowLineAsk	Gets/sets the value of "ShowLineAsk" property
ShowLastLine	Gets/sets the value of "ShowLastLine" property
ShowPeriodSep	Gets/sets the value of "ShowPeriodSep" property (show period separators)
ShowGrid	Gets/sets the value of "ShowGrid" property
ShowVolumes	Gets/sets the value of "ShowVolumes" property (color for volumes and levels of opened positions)
ShowObjectDescr	Gets/sets the value of "ShowObjectDescr" property (show description for graphic objects)
ShowDateScale	Sets the value of "ShowDateScale" property (date scale of the chart)
ShowPriceScale	Sets the value of "ShowPriceScale" property (price scale of the chart)
Color properties	
ColorBackground	Gets/sets the value of "ColorBackground" property (background color of the chart)
ColorForeground	Gets/sets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart)
ColorGrid	Gets/sets the value of "ColorGrid" property (color of the grid)
ColorBarUp	Gets/sets the value of "ColorBarUp" property (color for bull bars, their shadow and candle body outlines)
ColorBarDown	Gets/sets the value of "ColorBarDown" property (color for bear bars, their shadow and candle body outlines)
ColorCandleBull	Gets/sets the value of "ColorCandleBull" property (body color of the bull candle)
ColorCandleBear	Gets/sets the value of "ColorCandleBear" property (body color of the bear candle)
ColorChartLine	Gets/sets the value of "ColorChartLine" property (color for line chart and Doji candles)
ColorVolumes	Gets/sets the value of "ColorVolumes" property (color for volumes and levels of opened positions)

Access to protected data	
ColorLineBid	Gets/sets the value of "ColorLineBid" property (color of Bid line)
ColorLineAsk	Gets/sets the value of "ColorLineAsk" property (color of Ask line)
ColorLineLast	Gets/sets the value of "ColorLineLast" property (color of the last deal price line)
ColorStopLevels	Gets/sets the value of "ColorStopLevels" property (color of the SL and TP levels)
Read only properties	
VisibleBars	Gets total number of visible chart bars
WindowsTotal	Gets total number of chart windows, including the chart indicator subwindows
WindowIsVisible	Gets visibility flag of the specified chart subwindow
WindowHandle	Gets window handle of the chart (HWND)
FirstVisibleBar	Gets the number of the first visible bar of the chart
WidthInBars	Gets window width in bars.
WidthInPixels	Gets subwindow width in pixels.
HeightInPixels	Gets subwindow height in pixels.
PriceMin	Gets minimal price of the specified subwindow
PriceMax	Gets maximal price of the specified subwindow
Properties	
Attach	Assigns the current chart to the class instance
FirstChart	Assigns the first chart of the client terminal to the class instance
NextChart	Assigns the next chart of the client terminal to the class instance
Open	Opens chart with specified parameters and assign it to the class instance
Detach	Detaches chart from the class instance
Close	Closes chart assigned to the class instance
BringToTop	Show chart on top of other charts
EventObjectCreate	Sets a flag to send notifications of an event of new object creation on a chart

Access to protected data	
EventObjectDelete	Sets a flag to send notifications of an event of object deletion on a chart
Indicators	
IndicatorAdd	Adds an indicator with the specified handle into a specified chart subwindow
IndicatorDelete	Removes an indicator with a specified name from the specified chart subwindow
IndicatorsTotal	Returns the number of all indicators applied to the specified chart subwindow
IndicatorName	Returns the short name of the indicator on the specified chart subwindow
Navigation	
Navigate	Navigates the chart
Access to MQL5 API	
Symbol	Gets symbol of the chart
Period	Gets period of the chart
Redraw	Redraws chart, assigned to the class instance
GetInteger	The function returns the value of the corresponding object property
SetInteger	Sets new value for the property of the integer type
GetDouble	The function returns the value of the corresponding object property
SetDouble	Sets new value for the property of the double type
GetString	The function returns the value of the corresponding object property
SetString	Sets new value for the property of the string type
SetSymbolPeriod	Changes symbol and period of the chart assigned to the class instance
ApplyTemplate	Applies specified template to the chart
ScreenShot	Creates screenshot of the specified chart and saves it to .gif file
WindowOnDropped	Gets chart subwindow number corresponding to the object (expert or script) drop point
PriceOnDropped	Gets price coordinate corresponding to the object (expert or script) drop point

Access to protected data	
TimeOnDropped	Gets time coordinate corresponding to the object (expert or script) drop point
XOnDropped	Gets X coordinate corresponding to the object (expert or script) drop point
YOnDropped	Gets Y coordinate corresponding to the object (expert or script) drop point
Input/Output	
virtual Save	Saves object parameters to file
virtual Load	Loads object parameters from file
virtual Type	Gets graphic object type identifier

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

ChartID

Returns identifier of the chart.

```
long  ChartID() const
```

Return Value

Chart identifier assigned to the chart class instance. If there is no chart assigned, it returns -1.

Mode (Get Method)

Gets the value of "Mode" property (bars, candles, or line).

```
ENUM_CHART_MODE  Mode() const
```

Return Value

Value of "Mode" property of the object assigned to the class instance. If there is no chart assigned, it returns WRONG_VALUE.

Mode (Set Method)

Sets new value for "Mode" property (bars, candles, or line).

```
bool  Mode(
   ENUM_CHART_MODE  mode      // chart mode
   )
```

Parameters

mode

[in] Chart mode (candles, bars or line) of ENUM_CHART_MODE enumeration.

Return Value

true - successful, false - cannot change the mode.

Foreground (Get Method)

Gets the value of "Foreground" property.

```
bool  Foreground() const
```

Return Value

Value of "Foreground" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

Foreground (Set Method)

Sets new value for "Foreground" property.

```
bool  Foreground(
   bool  foreground      // flag value
   )
```

Parameters

foreground

[in] New value for "Foreground" property.

Return Value

true - successful, false - cannot change the property.

Shift (Get Method)

Gets the value of "Shift" property.

```
bool  Shift() const
```

Return Value

Value of "Shift" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

Shift (Set Method)

Sets new value for "Shift" property.

```
bool  Shift(
   bool  shift       // flag value
   )
```

Parameters

shift

[in] New value for "Shift" property.

Return Value

true - successful, false - cannot change the property.

ShiftSize (Get Method)

Gets the value of "ShiftSize" property (in percents).

```
double  ShiftSize() const
```

Return Value

Value of "ShiftSize" property of the chart assigned to the class instance. If there is no chart assigned, it returns EMPTY_VALUE.

ShiftSize (Set Method)

Sets new value for "Shift" property (in percents).

```
bool  ShiftSize(
   double  shift_size      // property value
   )
```

Parameters

shift_size

[in] New value for "ShiftSize" property (in percents).

Return Value

true - successful, false - cannot change the property.

AutoScroll (Get Method)

Gets the value of "AutoScroll" property.

```
bool  AutoScroll() const
```

Return Value

Value of "AutoScroll" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

AutoScroll (Set Method)

Sets new value for "AutoScroll" property.

```
bool  AutoScroll(
   bool  autoscroll      // flag value
   )
```

Parameters

autoscroll

[in] New value for "Autoscroll" property.

Return Value

true - successful, false - cannot change the property.

Scale (Get Method)

Gets the value of "Scale" property.

```
int  Scale() const
```

Return Value

Value of "Scale" property of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   int  scale      // property value
   )
```

Parameters

scale

[in] New value for "Scale" property.

Return Value

true - successful, false - cannot change the property.

ScaleFix (Get Method)

Gets the value of "ScaleFix" property (fixed chart scale or not).

```
bool  ScaleFix() const
```

Return Value

Value of "ScaleFix" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScaleFix (Set Method)

Sets new value for "ScaleFix" property.

```
bool  ScaleFix(
   bool  scale_fix      // property value
   )
```

Parameters

scale_fix

[in] New value for "ScaleFix" property.

Return Value

true - successful, false - cannot change the property.

ScaleFix_11 (Get Method)

Gets the value of "ScaleFix_11" property (chart scale is 1:1, or not).

```
bool  ScaleFix_11() const
```

Return Value

Value of "ScaleFix_11" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScaleFix_11 (Set Method)

Sets new value for "ScaleFix_11" property.

```
bool  ScaleFix_11(
   string  scale_11      // property value
   )
```

Parameters

scale_11

[in] New value for "ScaleFix_11" property.

Return Value

true - successful, false - cannot change the property.

FixedMax (Get Method)

Gets the value of "FixedMax" property (fixed maximal price).

```
double  FixedMax() const
```

Return Value

Value of "FixedMax" property of the chart assigned to the class instance. If there is no chart assigned, it returns EMPTY_VALUE.

FixedMax (Set Method)

Sets the new value for "FixedMax" property.

```
bool  FixedMax(
   double  max       // fixed maximum
   )
```

Parameters

max

[in] New value for "FixedMax" property.

Return Value

true - successful, false - cannot change the property.

FixedMin (Get Method)

Gets the value of "FixedMin" property (fixed minimal price).

```
double  FixedMin() const
```

Return Value

Value of "FixedMin" property of the chart assigned to the class instance. If there is no chart assigned, it returns EMPTY_VALUE.

FixedMin (Set Method)

Sets new value for "FixedMin" property.

```
bool  FixedMax(
   double  min      // fixed minimum
   )
```

Parameters

max

[in] New value for "FixedMin" property.

Return Value

true - successful, false - cannot change the property.

PointsPerBar (Get Method)

Gets the value of "PointsPerBar" property (in points per bar).

```
double  PointsPerBar() const
```

Return Value

Value of "PointsPerBar" property of the chart assigned to the class instance. If there is no chart assigned, it returns EMPTY_VALUE.

PointsPerBar (Set Method)

Sets new value for "PointsPerBar" property.

```
bool  PointsPerBar(
   double  ppb      // scale
   )
```

Parameters

ppb

[in] New scale (in points per bar).

Return Value

true - successful, false - cannot change the scale.

ScalePPB (Get Method)

Gets the value of "ScalePPB" property (scale is "point per bar" or not).

```
bool  ScalePPB() const
```

Return Value

Value of "ScalePPB" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ScalePPB (Set Method)

Sets new value for "ScalePPB" property.

```
bool  ScalePPB(
   bool  scale_ppb      // flag value
   )
```

Parameters

scale_ppb

[in] New value for "ScalePPB" property.

Return Value

true - successful, false - cannot change the property.

ShowOHLC (Get Method)

Gets the value of "ShowOHLC" property.

```
bool  ShowOHLC() const
```

Return Value

Value of "ShowOHLC" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowOHLC (Set Method)

Sets new value for "ShowOHLC" property.

```
bool  ShowOHLC(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowOHLC" property.

Return Value

true - success, false - cannot change the property.

ShowLineBid (Get Method)

Gets the value of "ShowLineBid" property.

```
bool  ShowLineBid() const
```

Return Value

Value of "ShowLineBid" property of the char, assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineBid (Set Method)

Sets new value for "ShowLineBid" property.

```
bool  ShowLineBid(
   bool  show       // property value
   )
```

Parameters

show

[in] New value for "ShowLineBid" property.

Return Value

true - successful, false - cannot change the property.

ShowLineAsk (Get Method)

Gets the value of "ShowLineAsk" property.

```
bool  ShowLineAsk() const
```

Return Value

Value of "ShowLineAsk" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineAsk (Set Method)

Sets new value for "ShowLineAsk" property.

```
bool  ShowLineAsk(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowLineAsk" property.

Return Value

true - successful, false - cannot change the property.

ShowLastLine (Get Method)

Gets the value of "ShowLastLine" property.

```
bool  ShowLastLine() const
```

Return Value

Value of "ShowLastLine" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowLastLine (Set Method)

Sets new value for "ShowLastLine" property.

```
bool  ShowLastLine(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowLastLine" property.

Return Value

true - successful, false - cannot change the property.

ShowPeriodSep (Get Method)

Gets the value of "ShowPeriodSep" property (show period separators).

```
bool  ShowPeriodSep() const
```

Return Value

Value of "ShowPeriodSep" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowPeriodSep (Set Method)

Sets new value for "ShowPeriodSep" property.

```
bool  ShowPeriodSep(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowPeriodSep" property.

Return Value

true - successful, false - cannot change the property.

ShowGrid (Get Method)

Gets the value of "ShowGrid" property.

```
bool  ShowGrid() const
```

Return Value

Value of "ShowGrid" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowGrid (Set Method)

Sets new value for "ShowGrid" property.

```
bool  ShowGrid(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowGrid" property.

Return Value

true - successful, false - cannot change the property.

ShowVolumes (Get Method)

Gets the value of "ShowVolumes" property.

```
bool  ShowVolumes() const
```

Return Value

Value of "ShowVolumes" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowVolumes (Set Method)

Sets new value for "ShowVolumes" property.

```
bool  ShowVolumes(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowVolumes" property.

Return Value

true - successful, false - cannot change the property.

ShowObjectDescr (Get Method)

Gets the value of "ShowObjectDescr" property (show description for graphic objects).

```
bool  ShowObjectDescr() const
```

Return Value

Value of "ShowObjectDescr" property of the chart assigned to the class instance. If there is no chart assigned, it returns false.

ShowObjectDescr (Set Method)

Sets new value for "ShowObjectDescr" property.

```
bool  ShowObjectDescr(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowObjectDescr" property.

Return Value

true - successful, false - cannot change the property.

ShowDateScale

Sets new value for "ShowDateScale" property.

```
bool  ShowDateScale(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowDateScale" property.

Return Value

true - successful, false - cannot change the property.

ShowPriceScale

Sets new value for "ShowPriceScale" property.

```
bool  ShowPriceScale(
   bool  show      // property value
   )
```

Parameters

show

[in] New value for "ShowPriceScale" property.

Return Value

true - successful, false - cannot change the property.

ColorBackground (Get Method)

Gets the value of "ColorBackground" property (background color of the chart).

```
color  ColorBackground() const
```

Return Value

Value of "ColorBackground" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBackground (Set Method)

Sets new value for "ColorBackground" property.

```
bool  ColorBackground(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New background color.

Return Value

true - successful, false - cannot change the color.

ColorForeground (Get Method)

Gets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart).

```
color  ColorForeground() const
```

Return Value

Value of "ColorForeground" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorForeground (Set Method)

Sets new value for "ColorForeground" property (for axes, scale, and OHLC string).

```
bool  ColorForeground(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color for axes, scale and OHLC string.

Return Value

true - successful, false - cannot change the color.

ColorGrid (Get Method)

Gets the value of "ColorGrid" property (color of the grid).

```
color  ColorGrid() const
```

Return Value

Value of "ColorGrid" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorGrid (Set Method)

Sets new value for "ColorGrid" property.

```
bool  ColorGrid(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New grid color.

Return Value

true - successful, false - cannot change the color.

ColorBarUp (Get Method)

Gets the value of "ColorBarUp" property (color for bullish bars, their shadow, and candle body outlines).

```
color  ColorBarUp() const
```

Return Value

Value of "ColorBarUp" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBarUp (Set Method)

Sets new value for "ColorBarUp" property.

```
bool  ColorBarUp(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color for bullish bars, their shadow and candle body outlines.

Return Value

true - successful, false - cannot change the color.

ColorBarDown (Get Method)

Gets the value of "ColorBarDown" property (color for bearish bars, their shadow, and candle body outlines).

```
color  ColorBarDown() const
```

Return Value

Value of "ColorBarDown" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBarDown (Set Method)

Sets new value for "ColorBarDown" property.

```
bool  ColorBarDown(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color for bearish bars, their shadow, and candle body outlines.

Return Value

true - successful, false - cannot change the color.

ColorCandleBull (Get Method)

Gets the value of "ColorCandleBull" property (body color of the bullish candle).

```
color  ColorCandleBull() const
```

Return Value

Value of "ColorCandleBull" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorCandleBull (Set Method)

Sets new value for "ColorCandleBull" property.

```
bool  ColorCandleBull(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the bullish candle body.

Return Value

true - successful, false - cannot change the color.

ColorCandleBear (Get Method)

Gets the value of "ColorCandleBear" property (body color of the bearish candle).

```
color  ColorCandleBear() const
```

Return Value

Value of "ColorCandleBear" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorCandleBear (Set Method)

Sets new value for "ColorCandleBear" property.

```
bool  ColorCandleBear(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the bearish candle body.

Return Value

true - successful, false - cannot change the color.

ColorChartLine (Get Method)

Gets the value of "ColorChartLine" property (color for line chart and Doji candles).

```
color  ColorChartLine() const
```

Return Value

Value of "ColorChartLine" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorChartLine (Set Method)

Sets new value for "ColorChartLine" property.

```
bool  ColorChartLine(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the chart lines and Doji candles.

Return Value

true - successful, false - cannot change the color.

ColorVolumes (Get Method)

Gets the value of "ColorVolumes" property (color for volumes and levels of opened positions).

```
color  ColorVolumes() const
```

Return Value

Value of "ColorVolumes" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorVolumes (Set Method)

Sets new value for "ColorVolumes" property.

```
bool  ColorVolumes(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the volumes and open position levels.

Return Value

true - successful, false - cannot change the color.

ColorLineBid (Get Method)

Gets the value of "ColorLineBid" property (color of Bid line).

```
color  ColorLineBid() const
```

Return Value

Value of "ColorLineBid" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineBid (Set Method)

Sets new value for "ColorLineBid" property.

```
bool  ColorLineBid(
   color  new_color       // color
   )
```

Parameters

new_color

[in] New color for Bid line.

Return Value

true - successful, false - cannot change the color.

ColorLineAsk (Get Method)

Gets the value of "ColorLineAsk" property (color of Ask line).

```
color  ColorLineAsk() const
```

Return Value

Value of "ColorLineAsk" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineAsk (Set Method)

Sets new value for "ColorLineAsk" property.

```
bool  ColorLineAsk(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color for Ask line.

Return Value

true - successful, false - cannot change the color.

ColorLineLast (Get Method)

Gets the value of "ColorLineLast" property (color of the last deal price line).

```
color  ColorLineLast() const
```

Return Value

Value of "ColorLineLast" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineLast (Set Method)

Sets new value for "ColorLineLast" property.

```
bool  ColorLineLast(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the last deal price line.

Return Value

true - successful, false - cannot change the color.

ColorStopLevels (Get Method)

Gets the value of "ColorStopLevels" property (color of the SL and TP levels).

```
color  ColorStopLevels() const
```

Return Value

Value of "ColorStopLevels" property of the chart assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorStopLevels (Set Method)

Sets new value for "ColorStopLevels" property.

```
bool  ColorStopLevels(
   color  new_color      // color
   )
```

Parameters

new_color

[in] New color of the Stop Loss and Take Profit price levels.

Return Value

true - successful, false - cannot change the color.

VisibleBars

Gets total number of visible chart bars.

```
int  VisibleBars() const
```

Return Value

Gets total number of visible bars of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

WindowsTotal

Gets total number of chart windows, including the chart indicator subwindows.

```
int  WindowsTotal() const
```

Return Value

Total number of windows, including the chart indicator subwindows, assigned to the class instance. If there is no chart assigned, it returns 0.

WindowIsVisible

Gets visibility flag of the specified chart subwindow.

```
bool  WindowIsVisible(
   int   num       // subwindow
   ) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Returns visibility flag of the specified chart subwindow assigned to the chart instance. If there is no chart assigned, it returns false.

WindowHandle

Gets window handle of the chart (HWND).

```
int  WindowHandle() const
```

Return Value

Window handle of the chart assigned to the chart instance. If there is no chart assigned, it returns INVALID_HANDLE.

FirstVisibleBar

Gets the number of the first visible bar of the chart.

```
int  FirstVisibleBar() const
```

Return Value

Number of the first visible bar of the chart assigned to the class instance. If there is no chart assigned, it returns -1.

WidthInBars

Gets chart width in bars.

```
int  WidthInBars() const
```

Return Value

Width in chart bars of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

WidthInPixels

Gets chart width in pixels.

```
int  WidthInPixels() const
```

Return Value

Width in pixels of the chart assigned to the chart instance. If there is no chart assigned, it returns 0.

HeightInPixels

Gets window height in pixels.

```
int  HeightInPixels(
   int  num       // subwindow
   ) const
```

Parameters

num

[in] Checked subwindow number (0 means main window).

Return Value

Window height in pixels of the chart assigned to the class instance. If there is no chart assigned, it returns 0.

PriceMin

Gets window minimal price.

```
double  PriceMin(
   int  num       // subwindow
   ) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Window minimal price value of the chart assigned to the class instance. If there is not chart assigned, it returns EMPTY_VALUE.

PriceMax

Gets window maximal price.

```
double  PriceMax(
   int  num       // subwindow
   ) const
```

Parameters

num

[in] Subwindow number (0 means main window).

Return Value

Window maximal price value of the chart assigned to the class instance. If there is not chart assigned, it returns EMPTY_VALUE.

Attach

Assigns the current chart to the class instance.

```
void  Attach()
```

Attach

Assigns the specified chart to the class instance.

```
void  Attach(
   long  chart      // chart identifier
   )
```

Parameters

chart

[in] Identifier of the assigned chart.

FirstChart

Assigns the first chart of the client terminal to the class instance.

```
void  FirstChart()
```

NextChart

Assigns the next chart (following the already assigned one) to the class instance.

```
void  NextChart()
```

Open

Opens chart with specified parameters and assigns it to the class instance.

```
long  Open(
   const string      symbol_name,      // symbol
   ENUM_TIMEFRAMES   timeframe         // period
   )
```

Parameters

symbol_name

[in] Chart symbol. NULL means the symbol of the current chart (to which an expert is attached).

timeframe

[in] Chart timeframe (from ENUM_TIMEFRAMES enumeration). 0 means the current timeframe.

Return Value

chart identifier.

Detach

Detaches chart from the class instance.

```
void  Detach()
```

Close

Closes chart assigned to the class instance.

```
void  Close()
```

BringToTop

Show chart on top of other charts.

```
bool  BringToTop() const
```

Return Value

true - successful, false - error.

EventObjectCreate

Sets a flag to send notifications of events of a graphical object creation.

```
bool  EventObjectCreate(
   bool  flag      // flag
   )
```

Parameters

flag

[in] New value of a flag to send notifications of events of a graphical object creation.

Return Value

true – successful, false - cannot change the flag.

EventObjectDelete

Sets a flag to send notifications of events of a graphical object deletion.

```
bool  EventObjectDelete(
   bool  flag        // flag
   )
```

Parameters

flag

[in] New value of a flag to send notifications of events of a graphical object deletion.

Return Value

true - successful, false - cannot change the flag.

IndicatorAdd

Adds an indicator with the specified handle into a specified chart window.

```
bool  IndicatorAdd(
   int    sub_win          // number of the subwindow
   int    handle           // handle of the indicator
   );
```

Parameters

sub_win

[in] The number of the chart subwindow. 0 means the main chart window. if the number of a non-existing window is specified, a new window will be created.

handle

[in] The handle of the indicator.

Return Value

The function returns true in case of success, otherwise it returns false. In order to obtain information about the error, call the GetLastError() function.

See also

IndicatorDelete(), IndicatorsTotal(), IndicatorName().

IndicatorDelete

Removes an indicator with a specified name from the specified chart window.

```
bool  IndicatorDelete(
   int             sub_win       // number of the subwindow
   const string    name          // short name of the indicator
   );
```

Parameters

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const name

[in] The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get the short name of an indicator, use the IndicatorName() function.

Return Value

Returns true in case of successful deletion of the indicator. Otherwise it returns false. To get error details, use the GetLastError() function.

Note

If two indicators with identical short names exist in the chart subwindow, the first one in a row will be deleted.

If other indicators on this chart are based on the values of the indicator that is being deleted, such indicators will also be deleted.

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

Deletion of an indicator from a chart does not mean that its calculation part will be deleted from the terminal memory. To release the indicator handle, use the IndicatorRelease() function.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the IndicatorDelete() function is identified by the short name.

See also

IndicatorAdd(), IndicatorsTotal(), IndicatorName(), iCustom(), IndicatorCreate(), IndicatorSetString().

IndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int  IndicatorsTotal(
   long    chart_id,    // chart identifier
   int     sub_win      // number of the subwindow
   );
```

Parameters

chart_id

[in] Chart identifier. 0 denotes the main chart.

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart window.

Return Value

The number of indicators in the specified chart window. To get error details, use the GetLastError() function.

Note

The function allows going searching through all the indicators attached to the chart. The number of all the windows of the chart can be obtained from the CHART_WINDOWS_TOTAL property using the GetInteger() function.

See also

IndicatorAdd(), IndicatorDelete(), IndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString().

IndicatorName

Returns the short name of the indicator by the index in the indicators list on the specified chart window.

```
string  IndicatorName(
   int    sub_win      // number of the subwindow
   int    index        // index of the indicator in the list of indicators added to the
   );
```

Parameters

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart window.

index

[in] Index of the indicator in the list of indicators. The numeration of indicators start with zero, i.e. the first indicator in the list has the 0 index. To obtain the number of indicators in the list, use the IndicatorsTotal() function.

Return Value

The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get error details, use the GetLastError() function.

Note

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the IndicatorDelete() function is identified by the short name.

See also

IndicatorAdd(), IndicatorDelete, IndicatorsTotal, iCustom(), IndicatorCreate(), IndicatorSetString().

Navigate

Shifts the chart.

```
bool  Navigate(
   ENUM_CHART_POSITION  position,      // position
   int                  shift=0        // shift
   )
```

Parameters

position

[in] Chart position (from ENUM_CHART_POSITION enumeration), relative to which a shift is performed.

shift=0

[in] Number of bars to shift.

Return Value

true - successful, false - cannot shift the chart.

Symbol

Gets chart symbol name.

```
string  Symbol() const
```

Return Value

Symbol name of the chart, assigned to the class instance. If there is no chart assigned, it returns "".

Period

Gets period of the chart.

```
ENUM_TIMEFRAMES  Period() const
```

Return Value

Period of the chart (from ENUM_TIMEFRAMES) assigned to the class instance. If there is no chart assigned, it returns 0.

Redraw

Redraws chart assigned to the class instance.

```
void  Redraw()
```

GetInteger

The function returns the value of the corresponding chart property. The chart property should be of the integer type. There are two variants of the function.

1. Immediately returns the property value.

```
long  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,          // property identifier
   int                          sub_window=0      // subwindow number
   ) const
```

2. If successful, puts the value of property to the specified variable of integer type, passed by reference as last parameter.

```
bool  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,        // property identifier
   int                          sub_window,     // subwindow number
   long&                        value           // link to the variable
   ) const
```

Parameters

prop_id

[in] Property identifier (ENUM_CHART_PROPERTY_INTEGER enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Link to the variable that receives the value of the requested property.

Return Value

Value of property of the chart assigned to the class instance. If there is not any chart assigned, it returns -1.

For the second variant, the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise it returns false. To read more about the error, call GetLastError().

SetInteger

Sets new value for the property of the integer type.

```
bool  SetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,      // property identifier
   long                         value         // value
   )
```

Parameters

prop_id

[in] Chart property identifier (from ENUM_CHART_PROPERTY_INTEGER enumeration).

value

[in] New value of the property.

Return Value

true - successful, false - cannot change the integer property.

GetDouble

The function returns the value of the corresponding chart property. The object property should be of the double type. There are two variants of the function.

1. Immediately returns the property value.

```
double  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,          // property identifier
   int                         sub_window=0      // subwindow number
   ) const
```

2. If successful, puts the value of property to the specified variable of double type, passed by reference as last parameter.

```
bool  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,         // property identifier
   int                         sub_window,      // subwindow number
   double&                     value            // link to the variable
   ) const
```

Parameters

prop_id

[in] Chart property identifier (from ENUM_CHART_PROPERTY_DOUBLE enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Variable of the double type that received the value of the requested property.

Return Value

Value of property of the chart assigned to the class instance. If there is not any chart assigned, it returns EMPTY_VALUE.

For the second variant the function, it returns true if the property value is received, otherwise returns false. To read more about the error, call GetLastError().

SetDouble

Sets new value for the chart property of the double type.

```
bool  SetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,      // property identifier
   double                      value         // new value
   )
```

Parameters

prop_id

[in] Chart property identifier (from ENUM_CHART_PROPERTY_DOUBLE enumeration).

value

[in] New value for the property.

Return Value

true - successful, false - cannot change the double property.

GetString

The function returns the value of the corresponding chart property. The chart property should be of the string type. There are two variants of the function.

1. Immediately returns the property value.

```
string  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id      // property identifier
   ) const
```

2. If successful, puts the value of property to the specified variable of string type, passed by reference as last parameter.

```
bool  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,     // property identifier
   string&                     value        // link to the variable
   ) const
```

Parameters

prop_id

[in] Chart property identifier (from ENUM_CHART_PROPERTY_STRING enumeration).

value

[in] Link to the variable that receives the value of the requested property.

Return Value

Value of a chart property assigned to the class instance. If there is no chart assigned, it returns "".

For the second variant, the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise it returns false. To read more about the error, call GetLastError().

SetString

Sets new value for the chart property of the string type.

```
bool  SetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,      // property identifier
   string                      value         // value
   )
```

Parameters

prop_id

[in] Chart property identifier (from ENUM_CHART_PROPERTY_STRING enumeration).

value

[in] New value for the property.

Return Value

true - successful, false - cannot change the string property.

SetSymbolPeriod

Changes symbol and period of the chart assigned to the class instance.

```
bool  SetSymbolPeriod(
   const string      symbol_name,      // symbol
   ENUM_TIMEFRAMES   timeframe         // period
   )
```

Parameters

symbol_name

[in] New chart symbol. NULL means the symbol of the current chart (to which an expert is attached).

timeframe

[in] New chart timeframe (from ENUM_TIMEFRAMES enumeration). 0 means the current chart timeframe.

Return Value

true - successful, false - cannot change the property.

ApplyTemplate

Applies specified template to the chart.

```
bool  ApplyTemplate(
   const string  filename      // template
   )
```

Parameters

filename

[in] File name of the template.

Return Value

true - successful, false - cannot apply the template.

ScreenShot

Creates a screenshot of the specified chart in its current state in .gif format.

```
bool  ScreenShot(
   string            filename,                    // file name
   int               width,                       // width
   int               height,                      // height
   ENUM_ALIGN_MODE   align_mode=ALIGN_RIGHT       // align type
   ) const
```

Parameters

filename

[in] File name for screenshot.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Align mode, if screenshot is narrow.

Return Value

true - successful, false - error.

WindowOnDropped

Gets chart subwindow number corresponding to the object (expert or script) drop point.

```
int  WindowOnDropped() const
```

Return Value

Chart subwindow number of the object drop point. 0 means main chart window.

PriceOnDropped

Gets price coordinate corresponding to the object (expert or script) drop point.

```
double  PriceOnDropped() const
```

Return Value

Price coordinate of the object drop point.

TimeOnDropped

Gets time coordinate corresponding to the object (expert or script) drop point.

```
datetime  TimeOnDropped() const
```

Return Value

Time coordinate of the object drop point.

XOnDropped

Gets X coordinate corresponding to the object (expert or script) drop point.

```
int  XOnDropped() const
```

Return Value

X coordinate of the object drop point.

YOnDropped

Gets Y coordinate corresponding to the object (expert or script) drop point.

```
int  YOnDropped() const
```

Return Value

Y coordinate of the object drop point.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int   file_handle       // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen(...) function.

Return Value

true - successful, false - error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int   file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen(...) function.

Return Value

true - successful, false - error.

Type

Returns type identifier.

```
virtual int  Type() const
```

Return Value

Type identifier (0x1111 for CChart).

Graphics

The graphics library contains classes and global functions for quick plotting of custom charts. The library provides convenient ready-made solutions for building axes, curves, as well as the methods for quick access to changing common properties of a custom chart.

To start working with the library, simply read the article Visualize this! MQL5 graphics library similar to 'plot' of R language.

The graphics library is placed to the Include\Graphics folder of the terminal's working directory.

Class	Description
CAxis	Class for working with coordinate axes
ColorGenerator	Class specifying the default color scheme
CCurve	Class for working with curves
CGraphic	Base class for creating custom charts

GraphPlot

Functions for quick curve plotting.

Version for plotting a single curve using Y coordinates.

```
string  GraphPlot(
   const double      &y[],                   // Y coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Note

Y array indices are used as X coordinates for the curve.

Version for plotting a single curve using X and Y coordinates

```
string  GraphPlot(
   const double      &x[],                   // X coordinates
   const double      &y[],                   // Y coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting two curves using X and Y coordinates

```
string  GraphPlot(
   const double      &x1[],                  // X coordinates
   const double      &y1[],                  // Y coordinates
   const double      &x2[],                  // X coordinates
   const double      &y2[],                  // Y coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting three curves using X and Y coordinates

```
string  GraphPlot(
   const double      &x1[],                  // X coordinates
   const double      &y1[],                  // Y coordinates
   const double      &x2[],                  // X coordinates
   const double      &y2[],                  // Y coordinates
   const double      &x3[],                  // X coordinates
   const double      &y3[],                  // Y coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting a curve using CPoint2D points coordinates

```
string  GraphPlot(
   const CPoint2D    &points[],              // curve coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting two curves using CPoint2D points coordinates

```
string  GraphPlot(
   const CPoint2D    &points1[],             // curve coordinates
   const CPoint2D    &points2[],             // curve coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting three curves using CPoint2D points coordinates

```
string  GraphPlot(
   const CPoint2D    &points1[],             // curve coordinates
   const CPoint2D    &points2[],             // curve coordinates
   const CPoint2D    &points3[],             // curve coordinates
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting a curve using the pointer to CurveFunction

```
string  GraphPlot(
   CurveFunction     function,               // pointer to the function
   const double      from,                   // initial value of the argument
   const double      to,                     // final value of the argument
   const double      step,                   // increment by the argument
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting two curves using the pointers to the CurveFunction functions

```
string  GraphPlot(
   CurveFunction     function1,              // pointer to the function
   CurveFunction     function2,              // pointer to the function
   const double      from,                   // initial value of the argument
   const double      to,                     // final value of the argument
   const double      step,                   // increment by the argument
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Version for plotting three curves using the pointers to the CurveFunction functions

```
string  GraphPlot(
   CurveFunction     function1,              // pointer to the function
   CurveFunction     function2,              // pointer to the function
   CurveFunction     function3,              // pointer to the function
   const double      from,                   // initial value of the argument
   const double      to,                     // final value of the argument
   const double      step,                   // increment by the argument
   ENUM_CURVE_TYPE   type=CURVE_POINTS       // curve type
   )
```

Parameters

&x[]

[in] X coordinates.

&y[]

[in] Y coordinates.

&x1[]

[in] X coordinates for the first curve.

&y1[]

[in] Y coordinates for the first curve.

&x2[]

[in] X coordinates for the second curve.

&y2[]

[in] Y coordinates for the second curve.

&x3[]

[in] X coordinates for the third curve.

&y3[]

[in] Y coordinates for the third curve.

&points[]

[in] Coordinates of the curve dots.

&points1[]

[in] Coordinates of the first curve dots.

&points2[]

[in] Coordinates of the second curve dots.

&points3[]

[in] Coordinates of the third curve dots.

function

[in] Pointer to the CurveFunction function.

function1

[in] Pointer to the first function.

function2

[in] Pointer to the second function.

function3

[in] Pointer to the third function.

from

[in] Corresponds to the first X coordinate.

to

[in] Corresponds to the last X coordinate.

step

[in] Parameter for calculating the X coordinates.

type=CURVE_POINTS

[in] Curve type.

Return Value

Name of a graphical resource.

CAxis

CAxis is an auxiliary graphics library class for working with the coordinate axes.

Description

The CAxis class receives and stores various parameters of the coordinate axes. The class implements the ability to auto scale the coordinate axes dynamically.

Declaration

```
class CAxis
```

Title

```
#include <Graphics\Axis.mqh>
```

Class methods

Method	Description
AutoScale	Get/set the auto-scaling flag
Min	Get/set the minimum axis value
Max	Get/set the maximum axis value
Step	Get the step value by axis
Name	Get/set the axis name
Color	Get/set the axis color
ValuesSize	Get/set the size of the axis numbers
ValuesWidth	Get/set the maximum displayed length of the axis numbers
ValuesFormat	Get/set the format of the axis numbers
ValuesDateTimeMode	Get the format of converting a date into a string.
ValuesFunctionFormat	Get the pointer to the function defining the format of displaying values on the axis.
ValuesFunctionFormatCBData	Get the pointer to the object that may contain additional data on converting axis values.
NameSize	Get/set the font size of the axis name
ZeroLever	Get/set the "zero lever" value
DefaultStep	Get/set the initial step value by axis
MaxLabels	Get/set the maximum amount of numbers on the axis

Method	Description
MinGrace	Get/set the "tolerance" value for the axis minimum
MaxGrace	Get/set the "tolerance" value for the axis maximum
SelectAxisScale	Auto scale the axis.

AutoScale (Get method)

Returns the flag defining the need for auto-scale.

```
bool  AutoScale()
```

Return Value

The flag value.

Note

true – perform auto-scaling.

false – do not perform auto-scaling.

AutoScale (Set method)

Sets the flag defining the need for auto-scale.

```
void  AutoScale(
   const bool  auto      // flag value
   )
```

Parameters

auto

[in]

Note

true – perform auto-scaling.

false – do not perform auto-scaling.

Min (Get method)

Returns the minimum axis value.

```
double  Min()
```

Return Value

Minimum axis value.

Min (Set method)

Sets the minimum axis value.

```
void  Min(
   const double  min       // minimum value
   )
```

Parameters

min

[in] Minimum value.

Max (Get method)

Returns the maximum axis value.

```
double  Max()
```

Return Value

Maximum axis value.

Max (Set method)

Sets the maximum axis value.

```
void  Max(
   const double  max      // maximum value
   )
```

Parameters

max

[in] Maximum axis value.

Step (Get method)

Return the step value by axis.

```
double  Step()
```

Return Value

Step value.

Name (Get method)

Returns the axis name.

```
string  Name()
```

Return Value

Axis name.

Name (Set method)

Sets the axis name.

```
void  Name(
   const string  name       // axis name
   )
```

Parameters

name

[in] Axis name.

Color (Get method)

Returns the axis color.

```
color  Color()
```

Return Value

Axis color.

Color (Set method)

Sets the axis color.

```
void  Color(
   const color  clr      // axis color
   )
```

Parameters

clr

[in] Axis color.

ValuesSize (Get method)

Returns the size of the axis numbers.

```
int  ValuesSize()
```

Return Value

Size of the axis numbers.

ValuesSize (Set method)

Sets the size of the axis numbers.

```
void  ValuesSize(
   const int  size       // size of the axis numbers
   )
```

Parameters

size

[in] Size of the axis numbers

ValuesWidth (Get method)

Returns the maximum allowed length in pixels for displaying the axis numbers.

```
int  ValuesWidth()
```

Return Value

Length of the axis numbers in pixels.

Note

If a length in pixels for a specified number exceeds the maximum allowed display length, it is truncated and ends in dots.

ValuesWidth (Set method)

Sets the maximum allowed length in pixels for displaying the axis numbers.

```
void  ValuesWidth(
   const int  width      // maximum allowed length in pixels
   )
```

Parameters

width

[in] Maximum allowed length of the axis numbers.

Note

If a length in pixels for a specified number exceeds the maximum allowed display length, it is truncated and ends in dots.

ValuesFormat (Get method)

Returns the format of the axis numbers.

```
string  ValuesFormat()
```

Return Value

Number format.

ValuesFormat (Set method)

Sets the format of the axis numbers.

```
void  ValuesFormat(
   const string  format      // format of the axis numbers
   )
```

Parameters

format

[in] Format of the axis numbers.

ValuesDateTimeMode (Get method)

Get the format of converting a date into a string.

```
int  ValuesDateTimeMode()
```

Return Value

Format of converting a date into a string.

ValuesDateTimeMode (Set method)

Set the format of converting a date into a string.

```
void  ValuesDateTimeMode(
   const int  mode      // format of converting a date into a string
   )
```

Parameters

mode

[in] Conversion format.

Note

Find out more about the formats of converting a date into a string in the TimeToString() function description.

ValuesFunctionFormat (Get method)

Get the pointer to the function defining the format of displaying values on the axis.

```
DoubleToStringFunction  ValuesFunctionFormat()
```

Return Value

Pointer to the function defining the format of displaying values on the axis.

ValuesFunctionFormat (Set method)

Set the pointer to the function defining the format of displaying values on the axis.

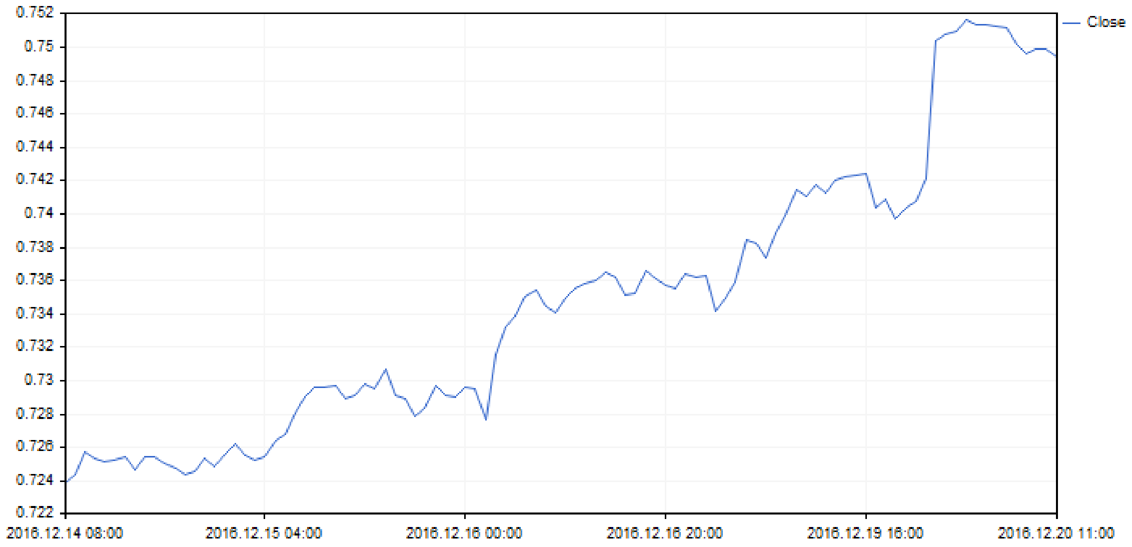
```
void  ValuesFunctionFormat(
   DoubleToStringFunction  func       // function for converting numerical values into
   )
```

Parameters

func

[in] Custom function for converting numerical values into a string.

Example:



The format of displaying X axis values has been changed using the following code:

```
//+------------------------------------------------------------------+
//|                                              DateAxisGraphic.mq5 |
//|                        Copyright 2016, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//--- array for store values
double arrX[];
double arrY[];
//+------------------------------------------------------------------+
//| Custom function for create values on X-axis                      |
//+------------------------------------------------------------------+
string TimeFormat(double x,void *cbdata)
  {
   return(TimeToString((datetime)arrX[ArraySize(arrX)-(int)x-1]));
  }
//+------------------------------------------------------------------+
void OnStart()
  {
   MqlRates rates[];
   CopyRates(Symbol(),Period(),0,100,rates);
   ArraySetAsSeries(rates,true);
   int size=ArraySize(rates);
   ArrayResize(arrX,size);
   ArrayResize(arrY,size);
   for(int i=0; i<size;++i)
     {
      arrX[i]=(double)rates[i].time;
      arrY[i]=rates[i].close;
     }
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"DateAxisGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"DateAxisGraphic");
     }
//--- create curve
   CCurve *curve=graphic.CurveAdd(arrY,CURVE_LINES);
//--- gets the X-axis
   CAxis *xAxis=graphic.XAxis();
//--- sets the X-axis properties
   xAxis.AutoScale(false);
   xAxis.Type(AXIS_TYPE_CUSTOM);
   xAxis.ValuesFunctionFormat(TimeFormat);
   xAxis.DefaultStep(20.0);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

ValuesFunctionFormatCBData (Get method)

Get the pointer to the object that may contain additional data on converting axis values.

```
void*  ValuesFunctionFormatCBData()
```

Return Value

The pointer to the object that may contain additional data on converting axis values.

ValuesFunctionFormatCBData (Set method)

Set the pointer to the class object that may contain additional data on converting axis values.

```
void  ValuesFunctionFormatCBData(
   void*   cbdata       // pointer to the class object
   )
```

Parameters

cbdata

[in] Pointer to any class object containing additional data on converting axis values

NameSize (Get method)

Returns the font size of the axis name.

```
int  NameSize()
```

Return Value

Font size of the axis name.

NameSize (Set method)

Sets the font size of the axis name.

```
void  NameSize(
   const int  size       // font size of the axis name
   )
```

Parameters

size

[in] Font size of the axis name.

ZeroLever (Get method)

Retuns the "zero lever" value.

```
double  ZeroLever()
```

Return Value

"Zero lever".

Note

The value is used to define when the axis scale range should be expanded to include a zero value.

ZeroLever (Set method)

Sets the "zero lever" value.

```
void  ZeroLever(
   const double  value      // "zero lever" value
   )
```

Parameters

value

[in] "Zero lever" value.

Note

The value is used to define when the axis scale range should be expanded to include a zero value.

DefaultStep (Get method)

Returns the initial step value by axis

```
double  DefaultStep()
```

Return Value

Step by axis.

DefaultStep (Set method)

Sets the initial step value by axis

```
void  DefaultStep(
   const double  value       // step by axis
   )
```

Parameters

value

[in] Initial step value by axis.

MaxLabels (Get method)

Returns the maximum allowed amount of numbers displayed on the axis.

```
double  MaxLabels()
```

Return Value

Maximum amount of numbers on the axis.

MaxLabels (Set method)

Sets the maximum allowed amount of numbers displayed on the axis.

```
void  MaxLabels(
   const double  value      // maximum number
   )
```

Parameters

value

[in] Maximum allowed amount of numbers displayed on the axis

MinGrace (Get method)

Returns the "tolerance" applied to the axis minimum.

```
double  MinGrace()
```

Return Value

"Tolerance" value for the axis minimum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

MinGrace (Set method)

Sets the "tolerance" applied to the axis minimum.

```
void  MinGrace(
   const double  value      // "tolerance" value
   )
```

Parameters

value

[in] "Tolerance" applied to the axis minimum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

MaxGrace (Get method)

Returns the "tolerance" applied to the axis maximum.

```
double  MaxGrace()
```

Return Value

"Tolerance" value for the axis maximum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MaxGrace is equal to 0.1, then 10% of the axis length (or 1.2) is added to the maximum value. As a result, the axis covers the interval from 4.0 to 17.2.

MaxGrace (Set method)

Sets the "tolerance" applied to the axis maximum.

```
void  MaxGrace(
   const double  value      // "tolerance" value
   )
```

Parameters

value

[in] "Tolerance" value applied to the axis maximum.

Note

This value is expressed as part of the overall axial length. For example, suppose that the axis values are located within 4.0 to 16.0, then its length is 12.0. If MinGrace is equal to 0.1, then 10% of the axis length (or 1.2) is subtracted from the minimum value. As a result, the axis covers the interval from 2.8 to 16.0.

SelectAxisScale

Auto scale the axis.

```
void  SelectAxisScale()
```

CColorGenerator

CColorGenerator class is an auxiliary graphics library class for working with the color palette.

Description

The CColorGenerator class contains the initial color palette used for curves by default (if a color is not specified by a user).

If all colors from the initial palette are used already, new colors are automatically generated and the palette is refilled.

Declaration

```
class CColorGenerator
```

Title

```
#include <Graphics\ColorGenerator.mqh>
```

Class methods

Method	Description
Next	Returns the next color from the palette
Reset	Resets the generator

Next

Returns the next color from the palette.

```
uint  Next()
```

Return Value

Color.

Note

If all colors from the palette have already been passed through, new colors are automatically generated in order to replace the old ones in the palette.

Reset

Resets the generator.

```
void  Reset()
```

CCurve

The CCurve class works with the properties of the curves generated on the chart.

Description

The CCurve class sets, installs and receives the coordinates and various properties of the curves when working with the CGraphic class.

There are three curve plotting modes: dots, lines and histogram. Separate parameters are implemented for each plotting mode in the class.

Declaration

```
class CCurve : public CObject
```

Title

```
#include <Graphics\Curve.mqh>
```

Inheritance hierarchy

CObject
    CCurve

Class methods

Method	Description
Type	Get the curve type
Name	Get the curve name
Color	Get the curve color
XMax	Get the maximum value of the X function
XMin	Get the minimum value of the X function
YMax	Get the maximum value of the Y function
YMin	Get the minimum value of the Y function
Size	Get the number of dots defining a curve
PointsSize	Get/set the linear size of dots defining a curve
PointsFill	Get/set the flag for filling dots defining a curve
PointsColor	Get/set the dot filling color
GetX	Gets X values of all curve dots to the array
GetY	Gets Y values of all curve dots to the array
LinesStyle	Get/set a line style when plotting a curve using lines

Method	Description
LinesIsSmooth	Get/set the smoothing flag when drawing using lines
LinesSmoothTension	Get/set the curve smoothing parameter when drawing using lines
LinesSmoothStep	Get/set the length of the approximating lines for smoothing when plotting by lines
LinesWidth	Get/set a line width when plotting a curve using lines
HistogramWidth	Get/set the width of columns when plotting using a histogram
CustomPlotCBData	Get/set the pointer to the object to be used in the custom curve plotting mode.
CustomPlotFunction	Get/set the pointer to the function implementing the custom curve plotting mode.
PointsType	Get/set the flag pointing at the type of dots used when plotting a dotted curve.
StepsDimension	Get/set the value indicating the dimension used in step-type curve rendering.
TrendLineCoefficients	Get/set trend line ratios for writing them into an array.
TrendLineColor	Get/set a color of a trend line for a curve.
TrendLineVisible	Get/set the trend line visibility flag.
Update	Update the curve coordinates.
Visible	Get/set the flag defining if a function is visible on the chart.

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Compare

Type

Return the curve type.

```
ENUM_CURVE_TYPE  Type()
```

Return Value

Curve type.

Name

Returns the curve name.

```
string  Name()
```

Return Value

Curve name.

Color

Returns the curve color.

```
uint  Color()
```

Return Value

Curve color.

XMax

Returns the maximum value of the X function (real numbers only).

```
double  XMax()
```

Return Value

Maximum real number among all the function arguments.

XMin

Returns the minimum value of the X function (real numbers only).

```
double  XMin()
```

Return Value

Minimum real number among all the function arguments.

YMax

Returns the maximum value of the Y function (real numbers only).

```
double  YMax()
```

Return Value

Maximum value of the Y function (real numbers only).

YMin

Returns the minimum value of the Y function (real numbers only).

```
double  YMin()
```

Return Value

Minimum value of the Y function (real numbers only).

Size

Returns the number of dots defining the curve.

```
int  Size()
```

Return Value

Number of dots defining the curve.

PointsSize (Get method)

Returns the linear size (in pixels) of dots used in plotting the curve.

```
int  PointsSize()
```

Return Value

Size of dots defining the curve in pixels.

PointsSize (Set method)

Sets the linear size (in pixels) of dots used in plotting the curve.

```
void  PointsSize(
   const int  size      // dot size in pixels
   )
```

Parameters

size

[in] Linear size (in pixels) of dots used in plotting the curve.

PointsFill (Get method)

Returns a flag determining if a filling for dots defining a curve should be performed.

```
bool  PointsFill ()
```

Return Value

The flag value.

Note

true – perform a filling

false – do not perform a filling

PointsFill (Set method)

Sets a flag determining if a filling for dots defining a curve should be performed.

```
void  PointsFill(
   const bool  fill        // flag value
   )
```

Parameters

fill

[in] Flag value.

Note

true – perform a filling

false – do not perform a filling

PointsColor (Get method)

Returns the dot filling color.

```
uint  PointsColor ()
```

Return Value

Color of filling dots defining the curve.

PointsColor (Set method)

Sets the dot filling color

```
void  PointsColor(
   const uint  clr      //dot filling color
   )
```

Parameters

clr

[in] Color of filling dots defining the curve.

GetX

Gets X values of all curve dots to the array.

```
void  GetX(
   double&  x[]       // array for writing X values
   )
```

Parameters

x[]

[out] Array for getting X values of all curve dots.

Note

Each curve dot is defined by a couple of X and Y values. These values are not coordinates in pixels for drawing in the CGraphic class.

GetY

Gets Y values of all curve dots to the array.

```
void  GetY(
   double&  y[]       // array for writing Y values
   )
```

Parameters

y[]

[out] Array for getting Y values of all curve dots.

Note

Each curve dot is defined by a couple of X and Y values. These values are not coordinates in pixels for drawing in the CGraphic class.

LinesStyle (Get method)

Returns a line style when plotting a curve using lines.

```
ENUM_LINE_STYLE  LinesStyle()
```

Return Value

Line style.

LinesStyle (Set method)

Sets a line style when plotting a curve using lines.

```
void  LinesStyle (
   ENUM_LINE_STYLE  style      // line style
   )
```

Parameters

style

[in] Line style.

LinesIsSmooth (Get method)

Returns a flag defining if smoothing should be done when plotting a curve by lines.

```
bool  LinesIsSmooth()
```

Return Value

Flag value

Note

true – perform smoothing

false – do not perform smoothing

LinesIsSmooth (Set method)

Sets a flag defining if smoothing should be done when plotting a curve by lines.

```
void  LinesIsSmooth(
   const bool  smooth      // flag value
   )
```

Parameters

smooth

[in] Flag value

Note

true – perform smoothing

false – do not perform smoothing

LinesSmoothTension (Get method)

Returns the curve smoothing parameter when drawing using lines.

```
double  LinesSmoothTension()
```

Return Value

Smoothing parameter value

Note

The 'tension' value is within the (0.0; 1.0] range.

LinesSmoothTension (Set method)

Sets the curve smoothing parameter when drawing using lines.

```
void  LinesSmoothTension(
   const double  tension      // parameter value
   )
```

Parameters

tension

[in] Smoothing parameter value.

Note

The 'tension' value is within the (0.0; 1.0] range.

LinesSmoothStep (Get method)

Returns the length of the approximating lines for smoothing when plotting by lines.

```
double  LinesSmoothStep()
```

Return Value

Length of approximating lines in pixels.

LinesSmoothStep (Set method)

Sets the length of the approximating lines for smoothing when plotting by lines.

```
void  LinesSmoothStep(
   const double  step      // line length
   )
```

Parameters

step

[in] Length of approximating lines

LinesEndStyle (Set method)

Get the flag indicating lines end plotting style when using lines to plot a curve.

```
ENUM_LINE_END  LinesEndStyle()
```

Return Value

A value of the flag indicating lines end plotting style when using lines to plot a curve.

LinesEndStyle (Get method)

Set the flag indicating lines end plotting style when using lines to plot a curve.

```
void  LinesEndStyle(
   ENUM_LINE_END  end_style      // flag value
   )
```

Parameters

end_style

[in] A value of the flag indicating line end plotting style when using lines to plot a curve.

LinesWidth (Get method)

Get lines width when plotting a curve using lines.

```
int  LinesWidth()
```

Return Value

Lines width.

LinesWidth (Set method)

Set lines width when plotting a curve using lines.

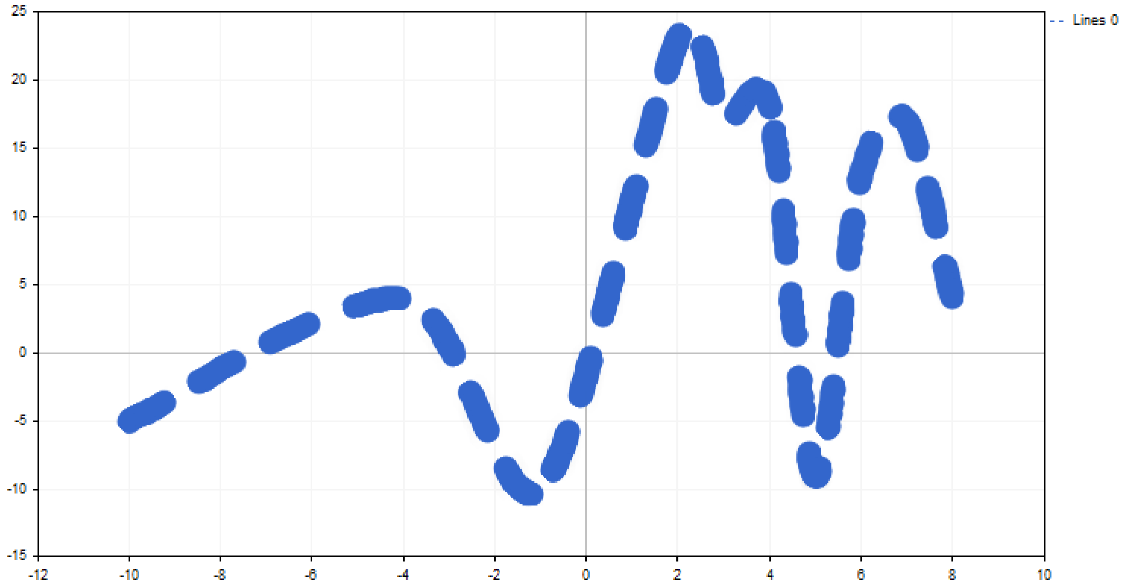
```
void  LinesWidth(
   const int  width      // lines width
   )
```

Parameters

width

[in] Lines width when plotting a curve using lines.

Example:



A line width has been changed using the following code:

```
//+------------------------------------------------------------------+
//|                                                CandleGraphic.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double x[]= { -100,-40,-10,20,30,40,50,60,70,80,120 };
   double y[]= { -5,4,-10,23,17,18,-9,13,17,4,9 };
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"ThickLineGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"ThickLineGraphic");
     }
//--- create curve
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_LINES);
//--- sets the curve properties
   curve.LinesSmooth(true);
   curve.LinesStyle(STYLE_DASH);
   curve.LinesEndStyle(LINE_END_ROUND);
   curve.LinesWidth(10);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

HistogramWidth (Get method)

Returns the width of columns when plotting using a histogram.

```
int  HistogramWidth()
```

Return Value

Column width in pixels.

HistogramWidth (Set method)

Sets the width of columns when plotting using a histogram.

```
void  HistogramWidth(
   const int   width       // column width
   )
```

Parameters

width

[in] Column width in pixels.

CustomPlotCBData (Get method)

Get the pointer to the object to be used in the custom curve plotting mode.

```
void*  CustomPlotCBData()
```

Return Value

Pointer to the object for the custom curve plotting mode.

CustomPlotCBData (Set method)

Set the pointer to the object to be used in the custom curve plotting mode.

```
void  CustomPlotCBData(
   void*  cbdata       // pointer to the object
   )
```

Parameters

cbdata

[in] The pointer to the object to be used in the custom curve plotting mode

CustomPlotFunction (Get method)

Get the pointer to the function implementing the custom curve plotting mode.

```
PlotFucntion  CustomPlotFunction()
```

Return Value

Pointer to the function implementing the custom curve plotting mode.

CustomPlotFunction (Set method)

Set the pointer to the function implementing the custom curve plotting mode.

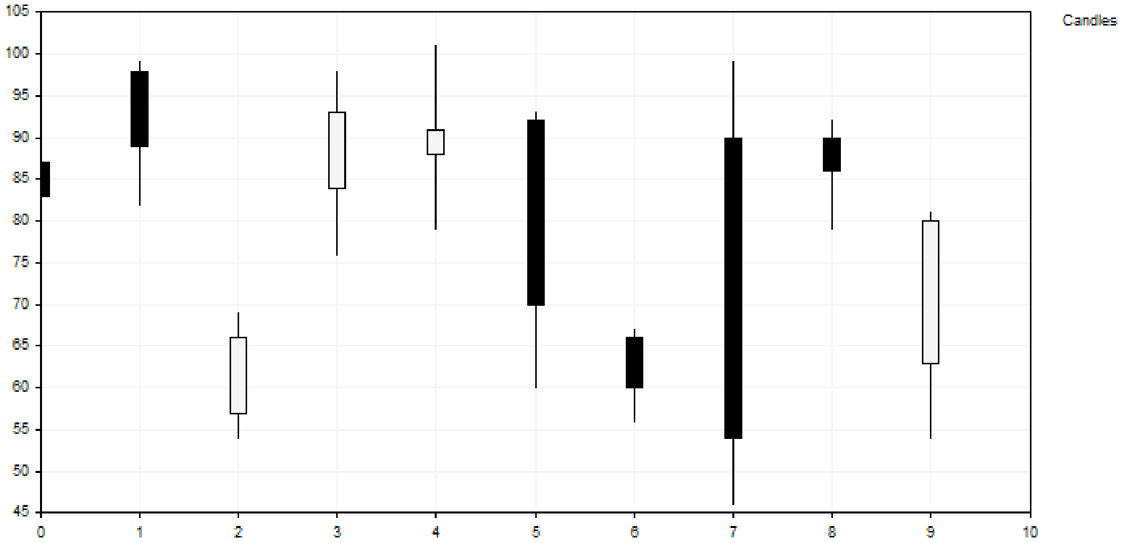
```
void  CustomPlotFunction(
   PlotFucntion  func      // pointer to the function
   )
```

Parameters

func

[in] Pointer to the function implementing the custom curve plotting mode

Example:



This curve consisting of bars is built using the following code:

```
//+------------------------------------------------------------------+
//|                                                CandleGraphic.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//| Class CCandle                                                    |
//| Usage: class to represent the candle                             |
//+------------------------------------------------------------------+
class CCandle: public CObject
  {
private:
   double            m_open;
   double            m_close;
   double            m_high;
   double            m_low;
   uint              m_clr_inc;
   uint              m_clr_dec;
   int               m_width;

public:
                     CCandle(const double open,const double close,const double high,co
                                                   const int width,const uint clr_
                    ~CCandle(void);
   double            OpenValue(void)            const { return(m_open);     }
   double            CloseValue(void)           const { return(m_close);    }
   double            HigthValue(void)           const { return(m_high);     }
   double            LowValue(void)             const { return(m_low);      }
   uint              CandleColorIncrement(void) const { return(m_clr_inc);  }
   uint              CandleColorDecrement(void) const { return(m_clr_dec);  }
   int               CandleWidth(void)          const { return(m_width);    }
  };
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CCandle::CCandle(const double open,const double close,const double high,const double 
                                 const int width,const uint clr_inc=0x000000,const ui
                                 m_open(open),m_close(close),m_high(high),m_low(low),
                                 m_clr_inc(clr_inc),m_clr_dec(clr_dec),m_width(width)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CCandle::~CCandle(void)
  {
  }
//+------------------------------------------------------------------+
//| Custom method for plot candles                                   |
//+------------------------------------------------------------------+
void PlotCandles(double &x[],double &y[],int size,CGraphic *graphic,CCanvas *canvas,vo
  {
//--- check obj
   CArrayObj *candles=dynamic_cast<CArrayObj*>(cbdata);
   if(candles==NULL || candles.Total()!=size)
      return;
//--- plot candles
   for(int i=0; i<size; i++)
     {
      CCandle *candle=dynamic_cast<CCandle*>(candles.At(i));
      if(candle==NULL)
         return;
      //--- primary calculate
      int xc=graphic.ScaleX(x[i]);
      int width_2=candle.CandleWidth()/2;
      int open=graphic.ScaleY(candle.OpenValue());
      int close=graphic.ScaleY(candle.CloseValue());
      int high=graphic.ScaleY(candle.HigthValue());
      int low=graphic.ScaleY(candle.LowValue());
      uint clr=(open<=close) ? candle.CandleColorIncrement() :  candle.CandleColorDec
      //--- plot candle
      canvas.LineVertical(xc,high,low,0x000000);
      //--- plot candle real body
      canvas.FillRectangle(xc+width_2,open,xc-width_2,close,clr);
      canvas.Rectangle(xc+width_2,open,xc-width_2,close,0x000000);
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int count=10;
   int width=10;
   double x[];
   double y[];
   ArrayResize(x,count);
   ArrayResize(y,count);
   CArrayObj candles();
   double max=0;
   double min=0;
//--- create values
   for(int i=0; i<count; i++)
     {
      x[i] = i;
      y[i] = i;
      //--- calculate values
      double open=MathRound(50.0+(MathRand()/32767.0)*50.0);
      double close=MathRound(50.0+(MathRand()/32767.0)*50.0);
      double high=MathRound(MathMax(open,close)+(MathRand()/32767.0)*10.0);
      double low=MathRound(MathMin(open,close) -(MathRand()/32767.0)*10.0);
      //--- find max and min
      if(i==0 || max<high)
         max=high;
      if(i==0 || min>low)
         min=low;
      //--- create candle
      CCandle *candle=new CCandle(open,close,high,low,width);
      candles.Add(candle);
     }
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"CandleGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"CandleGraphic");
     }
//--- create curve
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_CUSTOM,"Candles");
//--- sets the curve properties
   curve.CustomPlotFunction(PlotCandles);
   curve.CustomPlotCBData(GetPointer(candles));
```

```
//--- sets the graphic properties
   graphic.YAxis().Max((int)max);
   graphic.YAxis().Min((int)min);
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

PointsType (Get method)

Get the flag pointing at the type of dots used when plotting a dotted curve.

```
ENUM_POINT_TYPE  PointsType()
```

Return Value

A value of the flag indicating a type of dots.

PointsType (Set method)

Set the flag pointing at the type of dots used when plotting a dotted curve.

```
void  PointsType(
   ENUM_POINT_TYPE  type       // flag value
   )
```

Parameters

type

[in] A value of the flag pointing at the type of dots used when plotting a dotted curve.

StepsDimension (Get method)

Get the value indicating the dimension used in step-type curve rendering.

```
int  StepsDimension()
```

Return Value

Dimension used in step-type curve rendering.

StepsDimension (Set method)

Set the value indicating the dimension used in step-type curve rendering.

```
void  StepsDimension(
   const int  dimension      // dimension
   )
```

Parameters

dimension

[in] Dimension (0 or 1).

Note

0 — x (the horizontal line is followed by the vertical one).

1 — y (the vertical line is followed by the horizontal one).

TrendLineCoefficients (Get method)

Get trend line ratios for writing them into an array.

```
double&  TrendLineCoefficients()
```

Return Value

Trend line ratios.

TrendLineCoefficients (Set method)

Set trend line ratios for writing then into an array.

```
void  TrendLineCoefficients(
   double&  coefficients[]      // array for writing ratios
   )
```

Parameters

coefficients[]

[out] Array for writing ratios.

TrendLineColor (Get method)

Get a color of a trend line for a curve.

```
uint  TrendLineColor()
```

Return Value

Color of the trend line.

TrendLineColor (Set method)

Set a color of a trend line for a curve.

```
void  TrendLineColor(
   const uint  clr      // trend line color
   )
```

Parameters

clr

[in] Line color

TrendLineVisible (Get method)

Get the trend line visibility flag.

```
bool  TrendLineVisible()
```

Return Value

A value of the flag that specifies if a trend line is visible.

TrendLineVisible (Set method)

Set the trend line visibility flag.

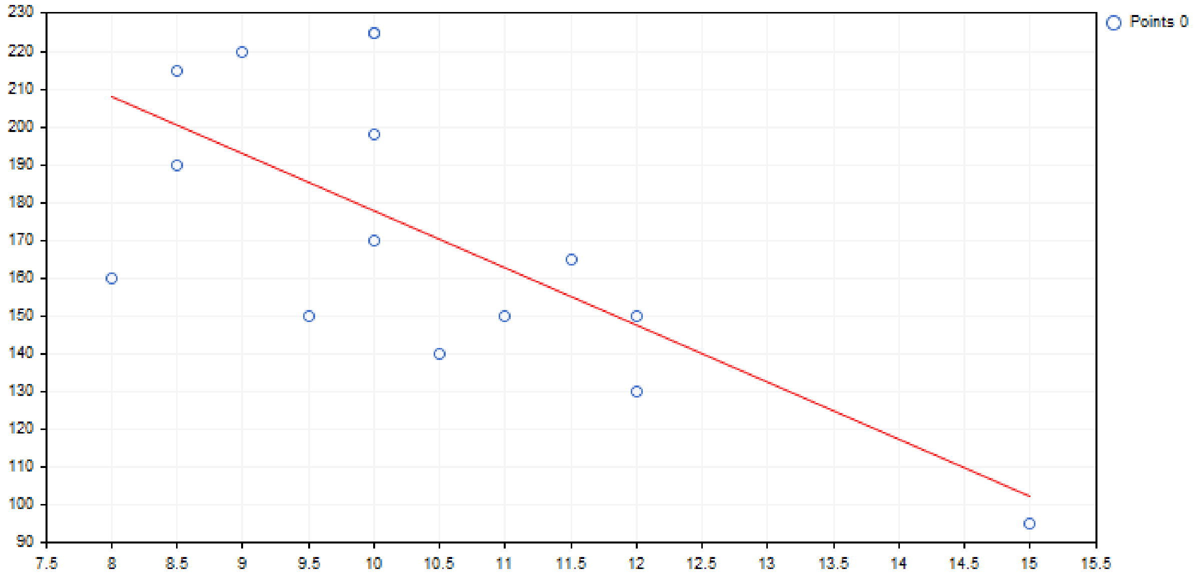
```
void  TrendLineVisible(
   const bool   visible       // flag value
   )
```

Parameters

visible

[in] A value of the trend line visibility flag.

Example:



Below is the code of the mentioned trend line and its plotting on the chart:

```
//+------------------------------------------------------------------+
//|                                            TrendLineGraphic.mq5 |
//|                             Copyright 2000-2024, MetaQuotes Ltd. |
//|                                            https://www.mql5.com |
//+------------------------------------------------------------------+
#include <Graphics\Graphic.mqh>
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double x[]={12.0,11.5,11.0,12.0,10.5,10.0,9.0,8.5,10.0,8.5,10.0,8.0,9.5,10.0,15.0};
   double y[]={130.0,165.0,150.0,150.0,140.0,198.0,220.0,215.0,225.0,190.0,170.0,160.
//--- create graphic
   CGraphic graphic;
   if(!graphic.Create(0,"TrendLineGraphic",0,30,30,780,380))
     {
      graphic.Attach(0,"TrendLineGraphic");
     }
//--- create curve
   CCurve *curve=graphic.CurveAdd(x,y,CURVE_POINTS);
//--- sets the curve properties
   curve.TrendLineVisible(true);
   curve.TrendLineColor(ColorToARGB(clrRed));
//--- plot
   graphic.CurvePlotAll();
   graphic.Update();
  }
```

Update

Update the curve coordinates.

The version for working by Y coordinate. Passed array indexes are used as X coordinates here.

```
void  Update(
   const double&  y[]    // Y coordinates
   )
```

This version uses X and Y coordinates.

```
void  Update(
   const double&  x[],      // X coordinates
   const double&  y[]       // Y coordinates
   )
```

The version for working with CPoint2D points.

```
void  Update(
   const CPoint2D&  points[]      // Curve coordinates
   )
```

The version for working with a pointer to the CurveFunction function.

```
void  Update(
   CurveFunction  function,      // pointer to the function describing a curve
   const double   from,          // initial value of the function argument
   const double   to,            // final value of the function argument
   const double   step           // argument increment
   )
```

Parameters

x[]

[in] X coordinates.

y[]

[in] Y coordinates.

points[]

[in] Curve coordinates.

function

[in] A pointer to the function describing a curve

from

[in] Initial value of the function argument

to

[in] End value of the function argument

step

[in] Argument increment

Visible (Get method)

Get the flag defining if a function is visible on the chart.

```
void  Visible(
   const bool   visible      //
   )
```

Return Value

A value of the flag defining a function visibility on the chart.

Visible (Set method)

Set the flag defining if a function is visible on the chart.

```
void  Visible(
   const bool   visible      // flag value
   )
```

Parameters

visible

[in] A value of the flag defining a function visibility on the chart.

CGraphic

CGraphic is a base class for creating custom charts.

Description

The CGraphic class provides numerous aspects of working with custom charts.

The class stores the main chart elements, sets their parameters and performs plotting.

Also, the class stores the curves for the chart and provides various display options.

Declaration

```
class CGraphic
```

Title

```
#include <Graphics\Graphic.mqh>
```

Class methods

Method	Description
Create	Create a graphical resource bound to a chart object
Destroy	Remove a chart and destroy a graphical resource
Update	Display implemented changes
ChartObjectName	Get the name of an object bound to a chart
ResourceName	Get the graphical resource name
XAxis	Get the pointer to the X axis
YAxis	Get the pointer to the Y axis
GapSize	Get/set the size of indents between the chart elements
BackgroundColor	Get/set a background color
BackgroundMain	Get/set a chart header
BackgroundMainSize	Get/set a sub-header font size
BackgroundMainColor	Get/set a chart header color
BackgroundSub	Get/set a sub-header
BackgroundSubSize	Get/set a sub-header font size
BackgroundSubColor	Get/set a chart sub-header color
GridLineColor	Get/set a grid line color

Method	Description
GridBackgroundColor	Get/set a grid background color
GridCircleRadius	Get/set the dot radius in the grid nodes
GridCircleColor	Get/set the dot color in the grid nodes
GridHasCircle	Get/set the dot plotting flag in the grid nodes
GridAxisLineColor	Get the value of a real chart axes color.
HistoryNameWidth	Get/set the maximum allowed length for displaying a curve name
HistoryNameSize	Get/set the font size of a curve name
HistorySymbolSize	Get/set a size of notational convention symbols
TextAdd	Add a text to the chart
LineAdd	Add a line to the chart
CurveAdd	Create and add a curve to the chart
CurvePlot	Plot a previously created curve by index
CurvePlotAll	Plot all previously created curves
CurveGetByIndex	Get a curve by a specified index
CurveGetByName	Get a curve by a specified name
CurveRemoveByIndex	Remove a curve by a specified index.
CurveRemoveByName	Remove a curve by a specified name.
CurvesTotal	Get the number of curves for the given chart.
MarksToAxisAdd	Add a scale mark to the chart axis
MajorMarkSize	Get/set the size of the scale's ticks on the chart axis
FontSet	Set the current font parameters
FontGet	Get the current font parameters
Attach	Get/set a graphical resource and bind it to the CGraphic class instance
CalculateMaxMinValues	Calculate (re-calculate) minimum and maximum chart values on both axes.
Height	Get a chart height in pixels.
IndentDown	Get/set a chart indent from the lower border.
IndentLeft	Get/set a chart indent from the left border.
IndentRight	Get/set a chart indent from the right border.

Method	Description
IndentUp	Get/set a chart indent from the upper border.
Redraw	Redraw the chart.
ResetParameters	Reset the chart redrawing parameters.
ScaleX	Scale the value by X axis.
ScaleY	Scale the value by Y axis.
SetDefaultParameters	Set the chart parameters to default values.
Width	Get the chart width in pixels.

Create

Creates a graphical resource bound to a chart object.

```
bool  Create(
   const long     chart,        // chart ID
   const string   name,         // name
   const int      subwin,       // sub-window index
   const int      x1,           // x1 coordinate
   const int      y1,           // y1 coordinate
   const int      x2,           // x2 coordinate
   const int      y2            // y1 coordinate
   )
```

Parameters

chart

[in] Chart ID.

name

[in] Name.

subwin

[in] Sub-window index.

x1

[in] X1 coordinate.

y1

[in] Y1 coordinate.

x2

[in] X2 coordinate.

y2

[in] Y2 coordinate.

Destroy

Removes a chart and destroys a graphical resource.

```
void  Destroy()
```

Update

Displays implemented changes.

```
void  Update(
   const bool   redraw=true       // flag
   )
```

Parameters

redraw=true

[in] Flag value

ChartObjectName

Gets the name of an object bound to a chart.

```
string  ChartObjectName()
```

Return Value

Name of an object bound to a chart.

ResourceName

Receives the name of a graphical resource.

```
string  ResourceName()
```

Return Value

Name of a graphical resource.

XAxis

Returns the pointer to the X axis.

```
CAxis  *XAxis()
```

Return Value

Pointer to the X axis.

YAxis

Returns the pointer to the Y axis.

```
CAxis  *YAxis()
```

Return Value

Pointer to the Y axis.

GapSize (Get method)

Returns the size of indents between the chart elements.

```
int  GapSize()
```

Return Value

Indent size in pixels.

GapSize (Set method)

Sets the size of indents between the chart elements.

```
void  GapSize(
   const int  size      // indent size
   )
```

Parameters

size

[in] Indent size in pixels.

BackgroundColor (Get method)

Returns the background color.

```
color  BackgroundColor()
```

BackgroundColor (Set method)

Sets the background color.

```
void  BackgroundColor(
   const color  clr      // background color
   )
```

Parameters

clr

[in] Background color.

BackgroundMain (Get method)

Returns a chart header

```
string  BackgroundMain()
```

BackgroundMain (Set method)

Sets a chart header text.

```
void  BackgroundMain(
   const string  main       // header text
   )
```

Parameters

main

[in] Chart header text

BackgroundMainSize (Get method)

Returns the header size.

```
int  BackgroundMainSize()
```

Return Value

Header font size.

BackgroundMainSize (Set method)

Sets the header size.

```
void  BackgroundMainSize(
   const int  size      // header size
   )
```

Parameters

size

[in] Header font size.

BackgroundMainColor (Get method)

Returns the header color.

```
color  BackgroundMainColor()
```

Return Value

Header color.

BackgroundMainColor (Set method)

Sets the header color.

```
void  BackgroundMainColor(
   const color  clr       // header color
   )
```

Parameters

clr

[in] Header color.

BackgroundSub (Get method)

Returns the sub-header.

```
string  BackgroundSub()
```

Return Value

Sub-header text.

BackgroundSub (Set method)

Sets the sub-header text.

```
void  BackgroundSub (Set method)(
   const string  sub      // sub-header text
   )
```

Parameters

sub

[in] Sub-header text.

BackgroundSubSize (Get method)

Returns the sub-header font size

```
int BackgroundSubSize()
```

BackgroundSubSize (Get method)

Sets the sub-header size

```
void  BackgroundSubSize(
   const int   size       // sub-header font size
   )
```

Parameters

size

[in] Sub-header font size

BackgroundSubColor (Get method)

Returns the sub-header color.

```
color  BackgroundSubColor()
```

BackgroundSubColor (Set method)

Sets the sub-header color.

```
void  BackgroundSubColor(
   const color  clr      // sub-header color
   )
```

Parameters

clr

[in] Sub-header color.

GridLineColor (Get method)

Returns the grid line color

```
color  GridLineColor()
```

Return Value

Grid line color.

GridLineColor (Set method)

Sets the grid line color.

```
void  GridLineColor(
   const color  clr      // line color
   )
```

Parameters

clr

[in] Grid line color.

GridBackgroundColor (Get method)

Returns the grid background color

```
color  GridBackgroundColor()
```

Return Value

Grid background color

GridBackgroundColor (Set method)

Sets the grid background color

```
void  GridBackgroundColor(
   const color  clr      // grid background color
   )
```

Parameters

clr

[in] Grid background color

GridCircleRadius (Get method)

Returns radius of dots in the grid nodes.

```
int  GridCircleRadius()
```

Return Value

Dot radius in pixels.

GridCircleRadius (Set method)

Sets the dot radius in the grid nodes

```
void  GridCircleRadius(
   const int  r       // radius
   )
```

Parameters

r

[in] Dot radius in pixels.

GridCircleColor (Get method)

Returns color of dots in the grid nodes.

```
color  GridCircleColor()
```

Return Value

Dot color.

GridCircleColor (Set method)

Sets the dot color in the grid nodes.

```
void  GridCircleColor(
   const color  clr       // dot color
   )
```

Parameters

clr

[in] Dot color.

GridHasCircle (Get method)

Returns the flag defining whether the dots in the grid nodes should be displayed.

```
bool  GridHasCircle()
```

Return Value

The flag value.

Note

true – display the dots

false – do not display the dots

GridHasCircle (Set method)

Sets the flag defining whether the dots in the grid nodes should be displayed.

```
void  GridHasCircle(
   const bool  has
   )
```

Parameters

has

[in] Flag value.

Note

true – display the dots

false – do not display the dots

GridAxisLineColor (Get method)

Get the value of a real chart axes color.

```
uint  GridAxisLineColor()
```

Return Value

Color of real chart axes.

GridAxisLineColor (Set method)

Set the value of a real chart axes color.

```
void  GridAxisLineColor(
   const uint  clr      // chart axes color
   )
```

Parameters

clr

[in] Color of real chart axes.

HistoryNameWidth (Get method)

Returns the maximum allowed length for displaying a curve name.

```
int  HistoryNameWidth()
```

Return Value

Maximum length in pixels.

Note

If the curve name exceeds the maximum allowed length, it is truncated and dots are added to its end.

HistoryNameWidth (Set method)

Sets the maximum allowed length for displaying a curve name.

```
void  HistoryNameWidth(
   const int  width      // maximum length
   )
```

Parameters

width

[in] Maximum length in pixels.

Note

If the curve name exceeds the maximum allowed length, it is truncated and dots are added to its end.

HistoryNameSize (Get method)

Returns the font size of the curve name.

```
int  HistoryNameSize()
```

Return Value

Font size of the curve name.

HistoryNameSize (Set method)

Sets the font size of the curve name.

```
void  HistoryNameSize (Set method)(
   const int  size       // name font size
   )
```

Parameters

size

[in] Name font size.

HistorySymbolSize (Get method)

Returns a size of a chart's notational convention symbols

```
int  HistorySymbolSize()
```

Return Value

Size of notational convention symbols

HistorySymbolSize (Set method)

Sets a size of a chart's notational convention symbols

```
void  HistorySymbolSize(
   const int  size      // symbol size
   )
```

Parameters

size

[in] Size of notational convention symbols.

TextAdd

Adds a text to a chart.

Version for working with X and Y coordinates

```
void  TextAdd(
   const int      x,                 // X coordinate
   const int      y,                 // Y coordinate
   const string   text,              // text
   const uint     clr,               // color
   const uint     alignment=0        // alignment
   )
```

Version for CPoint

```
void  TextAdd(
   const CPoint   &point,            // point coordinate
   const string   text,              // text
   const uint     clr,               // color
   const uint     alignment=0        // alignment
   )
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

&point

[in] Point coordinate.

text

[in] Text.

clr

[in] Color.

alignment=0

[in] Alignment.

LineAdd

Adds a line to a chart.

This version uses X and Y coordinates

```
void  LineAdd(
   const int    x1,          // x1 coordinate
   const int    y1,          // y1 coordinate
   const int    x2,          // x2 coordinate
   const int    y2,          // y2 coordinate
   const uint   clr,         // color
   const uint   style        // style
   )
```

Version for CPoint

```
void  LineAdd2(
   const CPoint  &point1,      // first point coordinate
   const CPoint  &point2,      // second point coordinate
   const uint    clr,          // color
   const uint    style         // style
   )
```

Parameters

x1

[in] X1 coordinate.

y1

[in] Y1 coordinate.

x2

[in] X2 coordinate.

y2

[in] Y2 coordinate.

&point1

[in] First point coordinate.

&point2

[in] Second point coordinate.

clr

[in] Color.

style

[in] Style.

CurveAdd

Create and add a new curve to the chart.

This version uses the Y coordinate (a curve color is set automatically)

```
CCurve* CurveAdd(
   const double      &y[],           // Y coordinates
   ENUM_CURVE_TYPE   type,           // curve type
   const string      name=NULL       // curve name
   )
```

Note

Y array indices are used as X coordinates for the curve.

This version uses the X and Y coordinates (a curve color is set automatically)

```
CCurve*  CurveAdd(
   const double      &x[],           // X coordinates
   const double      &y[],           // Y coordinates
   ENUM_CURVE_TYPE   type,           // curve type
   const string      name=NULL       // curve name
   )
```

The version for working with CPoint2D dots (curve color is set automatically)

```
CCurve*  CurveAdd(
   const CPoint2D    &points[],      // dot coordinates
   ENUM_CURVE_TYPE   type,           // curve type
   const string      name=NULL       // curve name
   )
```

Version for working with the pointer to the CurveFunction function (curve color is set automatically)

```
CCurve*  CurveAdd(
   CurveFunction     function,       // pointer to the function
   const double      from,           // initial value of the argument
   const double      to,             // final value of the argument
   const double      step,           // increment by the argument
   ENUM_CURVE_TYPE   type,           // curve type
   const string      name=NULL       // curve name
   )
```

Version for working by Y coordinate (a curve color is set by a user)

```
CCurve*  CurveAdd(
   const double      &y[],          // Y coordinates
   const uint        clr,           // curve color
   ENUM_CURVE_TYPE   type,          // curve type
   const string      name=NULL      // curve name
   )
```

Note

Y array indices are used as X coordinates for the curve.

This version uses the X and Y coordinates (a curve color is set by a user)

```
CCurve*  CurveAdd(
   const double      &x[],          // X coordinates
   const double      &y[],          // Y coordinates
   const uint        clr,           // curve color
   ENUM_CURVE_TYPE   type,          // curve type
   const string      name=NULL      // curve name
   )
```

The version for working with CPoint2D dots (curve color is set by a user)

```
CCurve*  CurveAdd(
   const CPoint2D    &points[],     // dot coordinates
   const uint        clr,           // curve color
   ENUM_CURVE_TYPE   type,          // curve type
   const string      name=NULL      // curve name
   )
```

Version for working with the pointer to the CurveFunction function (curve color is set by a user)

```
CCurve*  CurveAdd(
   CurveFunction     function,      // pointer to the function
   const double      from,          // initial value of the argument
   const double      to,            // final value of the argument
   const double      step,          // increment by the argument
   const uint        clr,           // curve color
   ENUM_CURVE_TYPE   type,          // curve type
   const string      name=NULL      // curve name
   )
```

Parameters

&x[]

[in] X coordinate.

&y[]

[in] Y coordinate.

&points[]

[in] Coordinates of dots.

function

[in] Pointer to the function.

from

[in] Initial value of the argument.

to

[in] Final value of the argument.

step

[in] Increment by the argument.

type

[in] Curve type.

name=NULL

[in] Curve name.

clr

[in] Curve color.

Return Value

Pointer to the created curve.

CurvePlot

Displays the previously created curve with a specified index.

```
bool  CurvePlot(
   const int  index       // index
   )
```

Parameters

index

[in] Curve index

Return Value

true - successful, otherwise - false.

CurvePlotAll

Displays all curves previously added to a chart.

```
bool  CurvePlotAll()
```

Return Value

true - successful, otherwise - false.

CurveGetByIndex

Gets the curve by a specified index.

```
CCurve*  CurveGetByIndex(
   const int  index       // curve index
   )
```

Parameters

index

[in] Curve index.

Return Value

Pointer to the curve with a specified index.

CurveGetByName

Gets a curve by a specified name.

```
CCurve*  CurveGetByName(
   const string  name       // curve name
   )
```

Parameters

name

[in] Curve name.

Return Value

Pointer to the first found curve having a specified name.

CurveRemoveByIndex

Remove a curve by a specified index.

```
bool  CurveRemoveByIndex(
   const int  index      // curve index
   )
```

Parameters

index

[in] Index of the curve to be removed.

Return Value

true — successful, otherwise — false.

CurveRemoveByName

Remove a curve by a specified name.

```
bool  CurveRemoveByName(
   const string  name       // curve name
   )
```

Parameters

name

[in] Name of the curve to be removed.

Return Value

true – successful, otherwise – false.

CurvesTotal

Get the number of curves for the given chart.

```
int  CurvesTotal()
```

Return Value

Number of curves.

Note

All curves on the current chart are considered regardless of drawing and visibility style.

MarksToAxisAdd

Add a scale mark (ticks) to the specified chart axis.

```
bool  MarksToAxisAdd(
   const double        &marks[],        // tick coordinates
   const int           mark_size,       // tick size
   ENUM_MARK_POSITION  position,        // tick location
   const int           dimension=0      // dimension
   )
```

Parameters

&marks[]

[in] Tick coordinates

mark_size

[in] Tick size

position

[in] Tick location

dimension=0

[in] 0 — adding to X axis,

1 — adding to Y axis

Return Value

true - successful, otherwise - false.

MajorMarkSize (Get method)

Returns the size of the scale's ticks on the coordinate axes.

```
int  MajorMarkSize()
```

MajorMarkSize (Set method)

Returns the size of the scale's ticks on the coordinate axes.

```
void  MajorMarkSize(
   const int  size       // tick size
   )
```

Parameters

size

[in] Tick size in pixels.

FontSet

Sets the current font parameters.

```
bool  FontSet(
   const string  name,          // name
   const int     size,          // size
   const uint    flags=0,       // flags
   const uint    angle=0        // angle
   )
```

Parameters

name

[in] Name.

size

[in] Size.

flags=0

[in] Flags.

angle=0

[in] Angle.

Return Value

true - successful, otherwise - false.

FontGet

Gets the current font parameters.

```
void  FontGet(
   string  &name,       // name
   int     &size,       // size
   uint    &flags,      // flags
   uint    &angle       // angle
   )
```

Parameters

&name

[out] Name.

&size

[out] Size.

&flags

[out] Flags.

&angle

[out] Angle.

Attach

The version for getting a graphical resource from the OBJ_BITMAP_LABEL object and binding it to the CGraphic class instance:

```
bool  Attach(
   const long     chart_id,      // chart ID
   const string   objname        // graphical object name
   )
```

The version for creating a graphical resource for the OBJ_BITMAP_LABEL object and binding it to the CGraphic class instance:

```
bool  Attach(
   const long     chart_id,      // chart ID
   const string   objname,       // graphical object name
   const int      width,         // image width
   const int      height         // image height
   )
```

Parameters

chart_id

[in] Chart ID.

objname

[in] Name of the graphical object.

width

[in] Image width in the resource.

height

[in] Image height in the resource.

Return Value

true – successful, false – failed to bind the object.

CalculateMaxMinValues

Calculate (re-calculate) minimum and maximum chart values on both axes.

```
void  CalculateMaxMinValues()
```

Height

Get a chart height in pixels.

```
int  Height()
```

Return Value

Chart height in pixels.

IndentDown (Get method)

Get a chart indent from the lower border.

```
int  IndentDown()
```

Return Value

Indent size in pixels.

IndentDown (Set method)

Set a chart indent from the lower border.

```
void  IndentDown(
   const int  down       // indent size
   )
```

Parameters

down

[in] Indent size in pixels.

IndentLeft (Get method)

Get a chart indent from the left border.

```
int  IndentLeft()
```

Return Value

Indent size in pixels.

IndentLeft (Set method)

Set a chart indent from the left border.

```
void  IndentLeft(
   const int  left       // indent size
   )
```

Parameters

left

[in] Indent size in pixels.

IndentRight (Get method)

Get a chart indent from the right border.

```
int  IndentRight()
```

Return Value

Indent size in pixels.

IndentRight (Set method)

Set a chart indent from the right border.

```
void  IndentRight(
   const int  right      // indent size
   )
```

Parameters

right

[in] Indent size in pixels.

IndentUp (Get method)

Get a chart indent from the upper border.

```
int  IndentUp()
```

Return Value

Indent size in pixels.

IndentUp (Set method)

Set a chart indent from the upper border.

```
void  IndentUp(
   const int  up       // indent size
   )
```

Parameters

up

[in] Indent value in pixels.

Redraw

Redraw the chart.

```
bool  Redraw(
   const bool   rescale=false      // flag value
   )
```

Parameters

rescale=false

[in] The flag indicating if a chart should be re-scaled.

Return Value

true - successful, otherwise - false.

ResetParameters

Reset the chart redrawing parameters.

```
void  ResetParameters()
```

ScaleX

Scale the value by X axis.

```
virtual int  ScaleX(
   double   x        // value by X axis
   )
```

Parameters

x

[in] Real value by X axis.

Return Value

A value in pixels.

Note

Scale a real value to pixels for displaying on the chart.

ScaleY

Scale the value by Y axis.

```
virtual int  ScaleY(
   double  y       // value by Y axis
   )
```

Parameters

y

[in] Real value by Y axis.

Return Value

A value in pixels.

Note

Scale a real value to pixels for displaying on the chart.

SetDefaultParameters

Set the chart parameters to default values.

```
void  SetDefaultParameters()
```

Width

Get the chart width in pixels.

```
int  Width()
```

Return Value

Chart width in pixels.

Technical Indicators and Timeseries

This section contains the technical details of the technical indicator and timeseries classes and description of the corresponding components of the standard MQL5 library.

The use of the technical indicator and timeseries classes will save time in developing applications (scripts, Expert Advisors).

The MQL5 Standard Library (in terms of technical indicators and timeseries) is located in the working directory of the terminal in the Include\Indicators folder.

Class/group	Description
Base classes	Group of base and auxiliary classes
Timeseries classes	Group of timeseries classes
Trend Indicators	Group of Trend indicator classes
Oscillators	Group of Oscillator indicator classes
Volume Indicators	Group of Volume indicator classes
Bill Williams Indicators	Group of Bill Williams indicator classes
Custom indicators	Custom indicator class

Base and Auxiliary Technical Indicator and Timeseries Classes

This section contains the technical details of base and auxiliary technical indicator and timeseries classes and description of the corresponding components of the Standard MQL5 library.

Class/group	Description
CSpreadBuffer	Historical spread buffer class
CTimeBuffer	Historical opening prices buffer class
CTickVolumeBuffer	Historical tick volumes buffer class
CRealVolumeBuffer	Historical real volumes buffer class
CDoubleBuffer	Base class of double type data buffer
COpenBuffer	Opening bar prices buffer class
CHighBuffer	High bar prices buffer class
CLowBuffer	Low bar prices buffer class
CCloseBuffer	Closing bar prices buffer class
CIndicatorBuffer	Technical indicator buffer class
CSeries	Base class for access to timeseries data
CPriceSeries	Base class for access to price data
CIndicator	Base class of technical indicator
CIndicators	Technical indicator and timeseries collection

Methods inherited from class CArrayObj

FreeMode, FreeMode, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

CSpreadBuffer

CSpreadBuffer is a class for simplified access to spreads of the bars in the history.

Description

The CSpreadBuffer class provides a simplified access to spreads of the bars in the history.

Declaration

```
class CSpreadBuffer: public CArrayInt
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayInt
      - CSpreadBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayInt

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Size

Sets buffer size.

```
void  Size(
   const int  size      // size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   const string           symbol,      // symbol
   const ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element.

```
int  At(
   const int  index       // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CTimeBuffer

CTimeBuffer is a class for simplified access to opening times of the bars in the history.

Description

The CTimeBuffer class provides a simplified access to opening times of the bars in the history.

Declaration

```
   class CTimeBuffer: public CArrayLong
```

Title

```
   #include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayLong
      - CTimeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Size

Sets buffer size.

```
void  Size(
   const int  size       // size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   const string            symbol,      // symbol
   const ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element.

```
long  At(
   const int   index       // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CTickVolumeBuffer

CTickVolumeBuffer is a class for simplified access to tick volumes of bars in the history.

Description

The CTickVolumeBuffer class provides a simplified access to tick volumes of bars in the history.

Declaration

```
class CTickVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayLong
      - CTickVolumeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Size

Sets buffer size.

```
void  Size(
   const int  size      // size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   const string            symbol,      // symbol
   const ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element.

```
long  At(
   const int  index       // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true – successful, false – cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CRealVolumeBuffer

CRealVolumeBuffer is a class for simplified access to real volumes of bars in the history.

Description

CTickVolumeBuffer class provides a simplified access to real volumes of bars in the history.

Declaration

```
class CRealVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayLong
      - CRealVolumeBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayLong

Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Size

Sets buffer size.

```
void  Size(
   const int  size      // size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   const string           symbol,      // symbol
   const ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element.

```
long  At(
   const int   index       // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CDoubleBuffer

CDoubleBuffer is a base class for simplified access to data buffers of double type.

Description

The CDoubleBuffer class provides a simplified access to the data of the buffer of double type.

Declaration

```
   class CDoubleBuffer: public CArrayDouble
```

Title

```
   #include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer

Direct descendants

CCloseBuffer, CHighBuffer, CIndicatorBuffer, CLowBuffer, COpenBuffer

Class Methods by Groups

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access	
At	Gets the buffer element
Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Size

Sets buffer size.

```
void  Size(
   const int  size      // size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   const string           symbol,      // symbol
   const ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element.

```
double  At(
   const int  index      // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

COpenBuffer

COpenBuffer is a class for simplified access to open prices of bars in the history.

Description

COpenBuffer class provides a simplified access to open prices of bars in the history.

Declaration

```
class COpenBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer
        - COpenBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Methods inherited from class CDoubleBuffer

Size, At, SetSymbolPeriod

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CHighBuffer

CHighBuffer is a class for simplified access to high prices of bars in the history.

Description

CHighBuffer class provides a simplified access to high prices of bars in the history.

Declaration

```
   class CHighBuffer: public CDoubleBuffer
```

Title

```
   #include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer
        - CHighBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Methods inherited from class CDoubleBuffer

Size, At, SetSymbolPeriod

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CLowBuffer

CLowBuffer is a class for simplified access to low prices of bars in the history.

Description

The CLowBuffer class provides a simplified access to low prices of bars in the history.

Declaration

```
class CLowBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer
        - CLowBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Methods inherited from class CDoubleBuffer

Size, At, SetSymbolPeriod

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true – successful, false – cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CCloseBuffer

CCloseBuffer is a class for simplified access to close prices of bars in the history.

Description

CCloseBuffer class provides a simplified access to close prices of bars in the history.

Declaration

```
   class CCloseBuffer: public CDoubleBuffer
```

Title

```
   #include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer
        - CCloseBuffer

Class Methods by Groups

Data Update	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update, Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Methods inherited from class CDoubleBuffer

Size, At, SetSymbolPeriod

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Return Value

true - successful, false - cannot update the buffer.

CIndicatorBuffer

CIndicatorBuffer is a class for simplified access to the data of the indicator's buffer.

Description

CIndicatorBuffer class provides the simplified access to the data buffer of technical indicator.

Declaration

```
   class CIndicatorBuffer: public CDoubleBuffer
```

Title

```
   #include <Indicators\Indicator.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayDouble
      - CDoubleBuffer
        - CIndicatorBuffer

Class Methods by Groups

Attributes	
Offset	Gets/sets offset of the buffer
Name	Gets/sets buffer name
Data Access	
At	Gets buffer's element
Data Update	
Refresh	Updates the buffer
RefreshCurrent	Updates only the current value

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayDouble

Delta, Type, Save, Load, Reserve, Resize, Shutdown, Add, AddArray, AddArray, Insert, InsertArray, InsertArray, AssignArray, AssignArray, At, operator, Minimum, Maximum, Update,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Shift, Delete, DeleteRange, CompareArray, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast, SearchLinear

Methods inherited from class CDoubleBuffer

Size, At, SetSymbolPeriod

Offset

Gets offset of the buffer.

```
int  Offset() const
```

Return Value

Buffer offset.

Offset

Sets offset of the buffer.

```
void  Offset(
   const int  offset       // offset
   )
```

Parameters

offset

[in] New buffer offset.

Name

Gets the name of the buffer.

```
string  Name() const
```

Return Value

Name of the buffer.

Name

Sets the name of the buffer.

```
void  Name(
   const string  name      // name
   )
```

Parameters

name

[in] New name of the buffer.

At

Gets buffer element.

```
double  At(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Return Value

Buffer element with the specified index.

Refresh

Updates the whole buffer.

```
bool  Refresh(
   const int  handle,      // indicator handle
   const int  num          // buffer number
   )
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Buffer index of the indicator.

Return Value

true - successful, false - cannot update the buffer.

RefreshCurrent

Updates the current (zeroth) buffer element.

```
bool  RefreshCurrent(
   const int  handle,      // handle of the indicator
   const int  num          // buffer number
   )
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Indicator buffer number.

Return Value

true - successful, false - cannot update the buffer.

CSeries

CSeries is a base class for an access to the timeseries data of the Standard Library.

Description

CSeries class provides the simplified access to all the MQL5 API general functions related to working with the series data for all its descendants (timeseries and indicator classes).

Declaration

```
class CSeries: public CArrayObj
```

Title

```
#include <Indicators\Series.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries

Direct descendants

CIndicator, CiRealVolume, CiSpread, CiTickVolume, CiTime, CPriceSeries

Class Methods by Groups

Attributes	
Name	Gets the name of timeseries or indicator
BuffersTotal	Gets the number of buffers of timeseries or indicator
Timeframe	Gets the timeframe flag of timeseries or indicator
Symbol	Gets the symbol of timeseries or indicator
Period	Gets the period of timeseries or indicator
RefreshCurrent	Sets/resets the flag of updating the current data
Data Access	
virtual BufferResize	Sets buffer size of timeseries or indicator
Data Update	
virtual Refresh	Update the data of timeseries or indicator
PeriodDescription	Transforms ENUM_TIMEFRAMES into a string

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Name

Gets the name of timeseries or indicator.

```
string  Name() const
```

Return Value

The name of timeseries or indicator.

BuffersTotal

Gets the number of buffers of timeseries or indicator.

```
int  BuffersTotal() const
```

Return Value

The number of buffers of timeseries or indicator.

Note

The timeseries has only one buffer.

Timeframe

Gets the timeframe flag of timeseries or indicator.

```
int  Timeframe() const
```

Return Value

The timeframe flag of timeseries or indicator.

Note

It is generated as the object visibility flags.

Symbol

Gets the symbol of timeseries or indicator.

```
string  Symbol() const
```

Return Value

The symbol of timeseries or indicator.

Period

Gets the period of timeseries or indicator.

```
ENUM_TIMEFRAMES  Period() const
```

Return Value

The period (value of ENUM_TIMEFRAMES enumeration) of timeseries or indicator.

RefreshCurrent

Sets a flag to constantly update the current values of timeseries or indicator.

```
string  RefreshCurrent(
   const bool   flag        // value
   )
```

Parameters

flag

[in] New flag.

Return Value

None.

BufferSize

Returns the amount of data available in timeseries buffer or indicator buffer.

```
int  BufferSize() const
```

Return Value

Amount of data available in timeseries buffer or indicator buffer.

BufferResize

Sets buffer size of timeseries or indicator.

```
virtual bool  BufferResize(
   const int  size       // size
   )
```

Parameters

size

[in] New size of the buffers.

Return Value

true - successful, otherwise - false.

Note

All the timeseries or indicator buffers have the same size.

Refresh

Updates the data of timeseries or indicator.

```
virtual void  Refresh(
   const int  flags       // flags
   )
```

Parameters

flags

[in] Timeframes to update (flag).

PeriodDescription

Gets the string representation of the specified ENUM_TIMEFRAMES enumeration.

```
string  PeriodDescription(
   const int  val=0      // value
   )
```

Parameters

val=0

[in] Value to convert.

Return Value

The string representation of the specified ENUM_TIMEFRAMES enumeration.

Note

If the value is not specified or equal to zero,the timeframe of timeseries or indicator is transformed into a string.

CPriceSeries

CPriceSeries is a base class for access to the price data.

Description

CSeries class provides the simplified access to MQL5 API general functions for working with price data to all its descendants.

Declaration

```
class CPriceSeries: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CPriceSeries

Direct descendants

CiClose, CiHigh, CiLow, CiOpen

Class Methods by Groups

Create	
virtual BufferResize	Sets size of the series buffer
Data Access	
virtual GetData	Gets the specified series buffer element
Data Update	
virtual Refresh	Updates timeseries data
Search for Extreme Values	
virtual MinIndex	Gets the index of minimal value in the specified range
virtual MinValue	Gets the minimal value in the specified range
virtual MaxIndex	Gets the index of maximal value in the specified range
virtual MaxValue	Gets the maximal value in the specified range

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

BufferResize

Sets the series buffer size.

```
virtual void  BufferResize(
   const int  size       // size
   )
```

Parameters

size

[in] New buffer size.

Return Value

true - successful, otherwise - false.

GetData

Gets the specified series buffer element.

```
double  GetData(
   const int  index      // index
   ) const
```

Parameters

index

[in] Index of a buffer element.

Return Value

The series buffer element, or EMPTY_VALUE.

Refresh

Updates the timeseries data

```
virtual void  Refresh(
   const int  flags=OBJ_ALL_PERIODS      // flags
   )
```

Parameters

flags=OBJ_ALL_PERIODS

[in] Timeframe flags.

MinIndex

Gets the index of minimal value in the specified range.

```
virtual int  MinIndex(
   const int  start,      // size
   const int  count       // number
   ) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

The index of minimal value of a series buffer in the specified range, or -1.

MinValue

Gets the minimal value in the specified range.

```
virtual double  MinValue(
   const int    start,      // size
   const int    count,      // number
   int&         index       // reference
   ) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for placing the found element's index value.

Return Value

The minimal value of the series buffer in the specified range, or EMPTY_VALUE.

Note

The index of the found element is stored by index reference.

MaxIndex

Gets the index of maximal value in the specified range.

```
virtual int  MaxIndex(
   const int  start,      // index
   const int  count       // number
   ) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

The index of the maximal value of the series buffer in the specified range, or -1.

MaxValue

Gets the maximal value in the specified range.

```
virtual double  MaxValue(
   const int    start,      // size
   const int    count,      // amount
   int&         index       // reference
   ) const
```

Parameters

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for placing the found element's index value.

Return Value

The maximal value of a series buffer in the specified range, or EMPTY_VALUE.

Note

The index of the found element is stored by index reference.

CIndicator

CIndicator is a base class for technical indicator classes of the MQL5 standard library.

Description

The CIndicator class provides the simplified access for all of its descendants to general MQL5 API technical indicator functions.

Declaration

```
class CIndicator: public CSeries
```

Title

```
#include <Indicators\Indicator.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator

Direct descendants

CiAC, CiAD, CiADX, CiADXWilder, CiAlligator, CiAMA, CiAO, CiATR, CiBands, CiBearsPower, CiBullsPower, CiBWMFI, CiCCI, CiChaikin, CiCustom, CiDEMA, CiDeMarker, CiEnvelopes, CiForce, CiFractals, CiFrAMA, CiGator, CiIchimoku, CiMA, CiMACD, CiMFI, CiMomentum, CiOBV, CiOsMA, CiRSI, CiRVI, CiSAR, CiStdDev, CiStochastic, CiTEMA, CiTriX, CiVIDyA, CiVolumes, CiWPR

Class Methods by Groups

Attributes	
Handle	Gets the indicator's handle.
Status	Gets the status of the indicator.
FullRelease	Sets a flag to release the handle.
Creation	
Create	Creates the indicator
BufferResize	Sets the new buffer sizes
Data Access	
GetData	Copies the data from the indicator buffer
Data Update Methods	
Refresh	Updates the indicator data

Attributes	
Finding Min/Max Values	
Minimum	Gets the index of minimal value in a specified range.
MinValue	Gets the minimal value in a specified range.
Maximum	Gets the index of maximal value in a specified range.
MaxValue	Gets the maximal value in a specified range.
Conversion of Enumerations	
MethodDescription	Converts ENUM_MA_METHOD into a string
PriceDescription	Converts ENUM_APPLIED_PRICE into a string
VolumeDescription	Converts ENUM_APPLIED_VOLUME into a string
Working with chart	
AddToChart	Adds an indicator to the chart
DeleteFromChart	Deletes an indicator from the chart

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Handle

Gets the indicator's handle.

```
int  Handle() const
```

Return Value

Handle of the indicator.

Status

Gets the status of the indicator.

```
string  Status() const
```

Return Value

The status of indicator creation.

FullRelease

Sets a flag to release the handle.

```
void  FullRelease(
   const bool   flag=true      // flag
   )
```

Parameters

flag

[in] New value of the handle release flag.

Create

Creates the indicator of the specified type with the specified parameters.

```
bool  Create(
   const string            symbol,          // symbol
   const ENUM_TIMEFRAMES   period,          // period
   const ENUM_INDICATOR    type,            // type
   const int               num_params,      // number of parameters
   const MqlParam&         params[]         // array of parameters
   )
```

Parameters

symbol

[in] Indicator symbol.

period

[in] Indicator timeframe (ENUM_TIMEFRAMES enumeration).

type

[in] Indicator type (ENUM_INDICATOR enumeration).

num_params

[in] Number of indicator's parameters.

params

[in] Reference to the parameters array for the indicator.

Return Value

true - successful, false - cannot create the indicator.

BufferResize

Sets the sizes of the indicator buffers.

```
virtual bool  BufferResize(
   const int  size      // size
   )
```

Parameters

size

[in] New buffer size.

Return Value

true - successful, otherwise - false.

Note

All the indicator buffers have the same size.

BarsCalculated

Returns the amount of calculated data for the indicator.

```
int  BarsCalculated() const;
```

Return Value

Returns the amount of calculated data in the indicator buffer, or -1 in the case of error (data is not calculated yet).

GetData

Gets the specified element of the specified buffer of the indicator. Refresh() should be called for working with recent data before using the method.

```
double  GetData(
   const int  buffer_num,       // buffer number
   const int  index             // index
   ) const
```

Parameters

buffer_num

[in] Indicator buffer number.

index

[in] Indicator buffer element index.

Return Value

value - success, EMPTY_VALUE - cannot receive the data.

GetData

Gets the data from the indicator's buffer by starting position and number.

```
int  GetData(
   const int       start_pos,       // position
   const int       count,           // number
   const int       buffer_num,      // buffer number
   double&         buffer[]         // array
   ) const
```

Parameters

start_pos

[in] Starting position of the indicator buffer.

count

[in] Number of indicator buffer elements.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified indicator buffer - success, otherwise -1.

GetData

Gets the data from the indicator buffer by start time and number.

```
int  GetData(
   const datetime  start_time,     // starting time
   const int       count,          // amount
   const int       buffer_num,     // buffer number
   double&         buffer[]        // array
   ) const
```

Parameters

start_time

[in] Indicator buffer element starting time.

count

[in] Number of indicator buffer elements.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified buffer, otherwise -1.

GetData

Gets the data from the indicator buffer by start and end time.

```
int  GetData(
   const datetime  start_time,     // start time
   const datetime  stop_time,      // end time
   const int       buffer_num,     // number of buffer
   double&         buffer[]        // array
   ) const
```

Parameters

start_time

[in] Indicator buffer initial element time.

stop_time

[in] Indicator buffer end element time.

buffer_num

[in] Number of the indicator buffer.

buffer

[in] Reference to the array for storing data.

Return Value

Number of the indicator values received from the specified buffer - success, otherwise -1.

Refresh

Updates the indicator data. It is recommended calling the method before using GetData().

```
virtual void  Refresh(
   int   flags=OBJ_ALL_PERIODS      // flags
   )
```

Parameters

flags=OBJ_ALL_PERIODS

[in] Timeframe update flags.

Minimum

Returns the index of minimal element of the specified buffer in a specified range.

```
int  Minimum(
   const int  buffer_num,      // buffer number
   const int  start,           // starting index
   const int  count            // number
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

Index of the minimal element of the specified buffer in a specified range.

MinValue

Returns the value of minimal element of the specified buffer in a specified range.

```
double  MinValue(
   const int    buffer_num,      // buffer number
   const int    start,           // starting index
   const int    count,           // number
   int&         index            // reference
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for storing the found element index value.

Return Value

The value of the minimal element of the specified buffer in a specified range.

Note

The index of the found element is stored by index reference.

Maximum

Returns the index of maximal element of the specified buffer in a specified range.

```
int  Maximum(
   const int  buffer_num,      // buffer number
   const int  start,           // starting index
   const int  count            // number
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

Return Value

Index of the maximal element of the specified buffer in a specified range.

MaxValue

Returns the value of maximal element of the specified buffer in a specified range.

```
double  MaxValue(
   const int    buffer_num,      // buffer number
   const int    start,           // starting index
   const int    count,           // number
   int&         index            // reference
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Search range initial index.

count

[in] Search range size (number of elements).

index

[out] Reference to the variable for storing the found element index value.

Return Value

The value of the maximal element of the specified buffer in a specified range.

Note

The index of maximal buffer element is stored by index reference.

MethodDescription

Converts ENUM_MA_METHOD enumeration value to a string.

```
string  MethodDescription(
   const int  val      // value
   ) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to ENUM_MA_METHOD enumeration value.

PriceDescription

Converts ENUM_APPLIED_PRICE enumeration value to a string.

```
string  PriceDescription(
   const int  val        // value
   ) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to ENUM_APPLIED_PRICE enumeration value.

VolumeDescription

Converts ENUM_APPLIED_VOLUME enumeration value to a string.

```
string  VolumeDescription(
   const int  val       // value
   ) const
```

Parameters

val

[in] Conversion value.

Return Value

The string corresponding to ENUM_APPLIED_VOLUME enumeration value.

AddToChart

Adds the indicator to the chart.

```
bool  AddToChart(
   const long   chart,      // chart ID
   const int   subwin       // subwindow index
   )
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow index.

Return Value

true - successful, false - cannot add the indicator to the chart.

DeleteFromChart

Deletes the indicator from the chart.

```
bool  DeleteFromChart(
   const long   chart,      // chart ID
   const int   subwin       // subwindow index
   )
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow index.

Return Value

true - successful, false - cannot remove the indicator from the chart.

CIndicators

The CIndicators is a class for collecting instances of timeseries and technical indicators classes.

Description

CIndicators class provides creation of the technical indicators class instances, their storage and management (data synchronization, handle and memory management).

Declaration

```
class CIndicators: public CArrayObj
```

Title

```
#include <Indicators\Indicators.mqh>
```

Class Methods by Groups

Create	
Create	Creates an indicator
Data Update	
Refresh	Updates indicator data

Create

Creates the indicator of the specified type with the specified parameters.

```
CIndicator*  Create(
   const string            symbol,      // symbol name
   const ENUM_TIMEFRAMES   period,      // period
   const ENUM_INDICATOR    type,        // type
   const int               count,       // number of parameters
   const MqlParam&         params       // parameters array
   )
```

Parameters

symbol

[in] Indicator symbol name.

period

[in] Indicator timeframe (ENUM_TIMEFRAMES enumeration value).

type

[in] Indicator type (ENUM_INDICATOR enumeration value).

count

[in] Number of parameters for the indicator.

params

[in] Reference to the parameters array for the indicator.

Return Value

Reference to the created indicator - successful, NULL - cannot create the indicator.

Refresh

Updates data for all timeseries and technical indicators in the collection.

```
int  Refresh()
```

Return Value

Updated timeframe flags (formed as object visibility flags).

Timeseries classes

This group of chapters contains technical details of timeseries classes of the MQL5 Standard Library and descriptions of all its key components.

Class	Description
CiSpread	Provides an access to spread historical data
CiTime	Provides an access to open times of the bars in the history
CiTickVolume	Provides an access to tick volumes of the bars in the history
CiRealVolume	Provides an access to real volumes of the bars in the history
CiOpen	Provides an access to open prices of the bars in the history
CiHigh	Provides an access to high prices of the bars in the history
CiLow	Provides an access to low prices of the bars in the history
CiClose	Provides an access to close prices of the bars in the history

CiSpread

CiSpread is a class designed for access to spreads of the bars in the history.

Description

CiSpread class provides an access to spread historical data.

Declaration

```
class CiSpread: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CiSpread

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets a series buffer size
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Create

Creates a timeseries with the specified parameters for access to the spreads history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

BufferResize

Sets size of the buffer series.

```
virtual void  BufferResize(
   int   size       // size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified element of timeseries buffer.

```
int  GetData(
   int  index       // index
   ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int   start_pos,      // position
   int   count,          // number
   int&  buffer          // array
   ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing the data.

Return Value

>=0 - successful, -1 - cannot receive the data.

GetData

Gets data from a timeseries buffer by initial time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   int&      buffer           // array
   ) const
```

Parameters

start_time

[in] Time of the timeseries buffer's initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by start and stop times.

```
int  GetData(
   datetime  start_time,      // starting time
   datetime  stop_time,       // stop time
   int&      buffer           // array
   ) const
```

Parameters

start_time

[in] Starting time of a timeseries buffer element.

stop_time

[in] Stop time of a timeseries buffer element.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive the data.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int   flags       // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiTime

CiTime is a class designed for access to open times of the bars in the history.

Description

CiTime class provides an access to open times of the bars in the history.

Declaration

```
class CiTime: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CiTime

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets timeseries buffer size
Data Access	
GetData	Gets the timeseries data
Data Update	
Refresh	Updates the timeseries data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Create

Creates a timeseries with the specified parameters for access to the opening times of the bars in the history.

```
bool  Create(
   string            symbol,      // symbol
   ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create timeseries.

BufferResize

Sets series buffer sizes.

```
virtual void  BufferResize(
   int   size       // size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified element of a timeseries buffer.

```
datetime  GetData(
   int  index       // index
   ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the data from a timeseries buffer by starting position and number.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   long&  buffer          // array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   long&     buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting and stop times.

```
int  GetData(
   datetime  start_time,      // starting time
   datetime  stop_time,       // stop time
   long&     buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer end element.

buffer

[in] Reference to the array for storing data

Return Value

>=0 - successful, -1 - cannot receive data.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int   flags       // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiTickVolume

CiTickVolume is a class designed for access to tick volumes of the bars in the history.

Description

CiTickVolume class provides an access to tick volumes of the bars in the history.

Declaration

```
class CiTickVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CiTickVolume

Class Methods by Groups

Create	
Create	Creates a timeseries
BufferResize	Sets buffer sizes
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Create

Creates a timeseries with the specified parameters for access to the tick volumes of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Timeseires symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

BufferResize

Sets size of the series buffer.

```
virtual void  BufferResize(
   int   size       // size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified timeseries buffer element.

```
long  GetData(
   int  index       // index
   ) const
```

Parameters

index

[in] Index of the buffer element.

Return Value

The timeseries buffer element, or 0.

GetData

Gets the data from the timeseries buffer by starting position and number.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   long& buffer           // array
   ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing the data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int        count,           // number
   long&      buffer           // array
   ) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the array for storing data.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting and stop times.

```
int  GetData(
   datetime  start_time,      // starting time
   datetime  stop_time,       // stop time
   long&     buffer           // array
   ) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

stop_time

[in] Time of the timeseries buffer end element.

buffer

[in] Reference to the array for storing data

Return Value

>=0 - successful, -1 - cannot receive data.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int   flags       // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiRealVolume

CiRealVolume is a class designed for access to real volumes of the bars in the history.

Description

CiRealVolume class provides an access to real volumes of the bars in the history.

Declaration

```
class CiRealVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CiRealVolume

Class Methods by Groups

Create	
Create	Creates a series
BufferResize	Sets buffer size
Data Access	
GetData	Gets the series data
Data Update	
Refresh	Updates the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Create

Creates a timeseries with the specified parameters for access to the real volumes of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

BufferResize

Sets a series buffer size.

```
virtual void  BufferResize(
   int   size       // size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified series buffer element.

```
datetime  GetData(
   int  index       // index
   ) const
```

Parameters

index

[in] Buffer element index.

Return Value

Series buffer element, or 0.

GetData

Gets the data from timeseries buffer by starting position and number.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   long&  buffer          // array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   long&     buffer           // array
   ) const
```

Parameters

start_time

[in] Time of the timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Return Value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int   flags       // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiOpen

CiOpen is a class designed for access to open prices of the bars in the history.

Description

CiOpen class provides an access to open prices of the bars in the history.

Declaration

```
class CiOpen: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CPriceSeries
          - CiOpen

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

Create

Creates a timeseries with the specified parameters for access to the open prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

GetData

Gets the element of timeseries by starting position and number.

```
int  GetData(
   int    start_pos,      // starting position
   int    count,          // number
   double& buffer         // array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   double&     buffer         // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from the timeseries buffer by starting and stop times.

```
int  GetData(
   datetime   start_time,      // starting time
   datetime   stop_time,       // stop time
   double&    buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

CiHigh

CiHigh is a class designed for access to high prices of the bars in the history.

Description

CiHigh class provides an access to high prices of the bars in the history.

Declaration

```
   class CiHigh: public CPriceSeries
```

Title

```
   #include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CPriceSeries
          - CiHigh

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets the series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

Create

Creates a timeseries with the specified parameters for access to the high prices of the bars in the history.

```
bool  Create(
   string            symbol,      // symbol
   ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create a timeseries.

GetData

Gets data from the timeseries buffer by starting position and number.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   double& buffer         // array
   ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   double&     buffer         // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting and stop times.

```
int  GetData(
   datetime   start_time,      // starting time
   datetime   stop_time,       // stop time
   double&    buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

CiLow

CiLow is a class designed for access to low prices of the bars in the history.

Description

CiLow class provides an access to low prices of the bars in the history.

Declaration

```
class CiLow: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CPriceSeries
          - CiLow

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets a series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

Create

Creates a timeseries with the specified parameters for access to the low prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   double& buffer         // array
   ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets the data from a timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   double&     buffer         // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from the timeseries buffer by starting and stop times.

```
int  GetData(
   datetime   start_time,      // starting time
   datetime   stop_time,       // stop time
   double&    buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array

Return Value

>=0 - successful, -1 - cannot receive data.

CiClose

CiClose is a class designed for access to close prices of the bars in the history.

Description

CiClose class provides an access to close prices of the bars in the history.

Declaration

```
class CiClose: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CPriceSeries
          - CiClose

Class Methods by Groups

Create	
Create	Creates a series
Data Access	
GetData	Gets series data

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Type, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CPriceSeries

BufferResize, MinIndex, MinValue, MaxIndex, MaxValue, GetData, Refresh

Create

Creates a timeseries with the specified parameters for access to the closing prices of the bars in the history.

```
bool  Create(
   string            symbol,      // symbol
   ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] Timeseries symbol.

period

[in] Timeseries timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the timeseries.

GetData

Gets data from a timeseries buffer by starting position and number.

```
int  GetData(
   int    start_pos,      // position
   int    count,          // number
   double& buffer         // array
   ) const
```

Parameters

start_pos

[in] Starting position of a timeseries buffer.

count

[in] Number of timeseries buffer elements.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from a timeseries buffer by starting time and number.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number
   double&     buffer         // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

count

[in] Number of a timeseries buffer element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

GetData

Gets data from a timeseries buffer by starting and stop times.

```
int  GetData(
   datetime   start_time,      // starting time
   datetime   stop_time,       // stop time
   double&    buffer           // array
   ) const
```

Parameters

start_time

[in] Time of a timeseries buffer initial element.

stop_time

[in] Time of a timeseries buffer last element.

buffer

[in] Reference to the data storage array.

Return Value

>=0 - successful, -1 - cannot receive data.

Main and Auxiliary Classes of Technical Indicators and Timeseries

This group of chapters contains technical details of the main and auxiliary classes of technical indicators and timeseries, as well as descriptions of the appropriate components of the MQL5 Standard Library.

Class/group	Description
CiADX	Average Directional Index
CiADXWilder	Average Directional Index by Welles Wilder
CiBands	Bollinger Bands®
CiEnvelopes	Envelopes
CiIchimoku	Ichimoku Kinko Hyo
CiMA	Moving Average
CiSAR	Parabolic Stop And Reverse System
CiStdDev	Standard Deviation
CiDEMA	Double Exponential Moving Average
CiTEMA	Triple Exponential Moving Average
CiFrAMA	Fractal Adaptive Moving Average
CiAMA	Adaptive Moving Average
CiVIDyA	Variable Index Dynamic Average

CiADX

CiADX is a class intended for using the Average Directional Index technical indicator.

Description

CiADX class provides the creation, configuration, and access to the data of the Average Directional Index indicator.

Declaration

```
   class CiADX: public CIndicator
```

Title

```
   #include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiADX

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element of the main line
Plus	Returns the buffer element of the +DI line
Minus	Returns the buffer element of the -DI line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,         // symbol
   ENUM_TIMEFRAMES    period,         // period
   int                ma_period       // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element of the main line by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Main line buffer element index.

Return Value

Main line buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double  Plus(
   int  index       // index
   )
```

Parameters

index

[in] +DI line buffer element index.

Return Value

The buffer element of the +DI line by the specified index, or EMPTY_VALUE if there is no correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double  Minus(
   int   index       // index
   )
```

Parameters

index

[in] -DI line buffer element index.

Return Value

The buffer element of the -DI line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ADX for CiADX).

CiADXWilder

CiADXWilder is a class intended for using the technical indicator Average Directional Index by Welles Wilder.

Description

CiADXWilder class provides the creation, configuration, and access to the data of the Average Directional Index by Welles Wilder.

Declaration

```
   class CiADXWilder: public CIndicator
```

Title

```
   #include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiADXWilder

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Plus	Returns the buffer data of the +DI line
Minus	Returns the buffer data of the -DI line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,        // symbol
   ENUM_TIMEFRAMES  period,        // period
   int              ma_period      // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element of the main line by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Main line buffer element index.

Return Value

Main line buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double  Plus(
   int  index       // index
   )
```

Parameters

index

[in] +DI line buffer element index.

Return Value

The buffer element of the +DI line by the specified index, or EMPTY_VALUE if there is no correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double  Minus(
   int  index       // index
   )
```

Parameters

index

[in] -DI line buffer element index.

Return Value

The buffer element of the -DI line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ADXW for CiADXWilder).

CiBands

CiBands is a class intended for using the Bollinger Bands® technical indicator.

Description

CiBands class provides the creation, configuration, and access to the data of the Bollinger Bands indicator.

Declaration

```
   class CiBands: public CIndicator
```

Title

```
   #include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiBands

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Base	Returns the buffer element of the base line
Upper	Returns the buffer element of the upper line
Lower	Returns the buffer element of the lower line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift.

```
int  MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Deviation

Returns the deviation.

```
double  Deviation() const
```

Return Value

Returns the deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,         // symbol
   ENUM_TIMEFRAMES   period,         // period
   int               ma_period,      // averaging period
   int               ma_shift,       // shift
   double            deviation,      // deviation
   int               applied         // applied price, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift of the indicator.

deviation

[in] Deviation.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Base

Returns the buffer element of the base line by the specified index.

```
double  Base(
   int  index       // index
   )
```

Parameters

index

[in] Base line buffer element index.

Return Value

The buffer element of the base line by the specified index, or EMPTY_VALUE if there is no correct data.

Upper

Returns the buffer element of the upper line by the specified index.

```
double  Upper(
   int  index       // index
   )
```

Parameters

index

[in] Upper line buffer element index.

Return Value

The buffer element of the upper line of the specified index, or EMPTY_VALUE if there is no correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double  Lower(
   int  index       // index
   )
```

Parameters

index

[in] Lower line buffer element index.

Return Value

The buffer element of the lower line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_BANDS for CiBands).

CiEnvelopes

CiEnvelopes is a class intended for using the Envelopes technical indicator.

Description

CiEnvelopes class provides the creation, configuration, and access to the data of the Envelopes indicator.

Declaration

```
class CiEnvelopes: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiEnvelopes

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Upper	Returns the buffer data of the upper line
Lower	Returns the buffer data of the lower line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation (ENUM_MA_METHOD enumeration value).

Deviation

Returns the value of deviation.

```
double  Deviation() const
```

Return Value

Returns the value of deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,         // symbol
   ENUM_TIMEFRAMES    period,         // period
   int                ma_period,      // averaging period
   int                ma_shift,       // shift
   ENUM_MA_METHOD     ma_method,      // averaging method
   int                applied,        // price type, handle
   double             deviation       // deviation
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Price axis shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Object (price type or handle) to apply.

deviation

[in] Deviation.

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the buffer element of the upper line by the specified index.

```
double  Upper(
   int  index       // index
   )
```

Parameters

index

[in] Upper line buffer element index.

Return Value

The buffer element of the upper line by the specified index, or EMPTY_VALUE if there is no correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double  Lower(
   int  index        // index
   )
```

Parameters

index

[in] Lower line buffer element index.

Return Value

The buffer element of the lower line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ENVELOPES for CiEnvelopes).

CiIchimoku

CiIchimoku is a class intended for using the Ichimoku Kinko Hyo technical indicator.

Description

CiIchimoku class provides the creation, setup, and access to the data of the Ichimoku Kinko Hyo indicator.

Declaration

```
class CiIchimoku: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiIchimoku

Class Methods by Groups

Attributes	
TenkanSenPeriod	Returns the TenkanSen period
KijunSenPeriod	Returns the KijunSen period
SenkouSpanBPeriod	Returns the SenkouSpanB period
Create	
Create	Creates the indicator
Data Access	
TenkanSen	Returns the buffer element of the TenkanSen line
KijunSen	Returns the buffer element of the KijunSen line
SenkouSpanA	Returns the buffer element of the SenkouSpanA line
SenkouSpanB	Returns the buffer element of the SenkouSpanB line
ChinkouSpan	Returns the buffer element of the ChikouSpan line
Input/output	

Attributes	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

TenkanSenPeriod

Returns the TenkanSen period.

```
int  TenkanSenPeriod() const
```

Return Value

Returns the TenkanSen period, defined at the indicator creation.

KijunSenPeriod

Returns the KijunSen period.

```
int  KijunSenPeriod() const
```

Return Value

Returns the KijunSen period, defined at the indicator creation.

SenkouSpanBPeriod

Returns the SenkouSpanB period.

```
int  SenkouSpanBPeriod() const
```

Return Value

Returns the SenkouSpanB period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,            // symbol
   ENUM_TIMEFRAMES    period,            // period
   int                tenkan_sen,        // period of TenkanSen
   int                kijun_sen,         // period of KijunSen
   int                senkou_span_b      // period of SenkouSpanB
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

tenkan_sen

[in] Period of TenkanSen.

kijun_sen

[in] Period of KijunSen.

senkou_span_b

[in] Period of SenkouSpanB.

Return Value

true - successful, false - cannot create the indicator.

TenkanSen

Returns the buffer element of the TenkanSen line by the specified index.

```
double  TenkanSen(
   int  index       // index
   )
```

Parameters

index

[in] TenkanSen line buffer element index.

Return Value

The buffer element of the TenkanSen line by the specified index, or EMPTY_VALUE if there is no correct data.

KijunSen

Returns the buffer element of the KijunSen line by the specified index.

```
double  KijunSen(
   int  index      // index
   )
```

Parameters

index

[in] KijunSen line buffer element index.

Return Value

The buffer element of the KijunSen line of the specified index, or EMPTY_VALUE if there is no correct data.

SenkouSpanA

Returns the buffer element of the SenkouSpanA line by the specified index.

```
double  SenkouSpanA(
   int  index       // index
   )
```

Parameters

index

[in] SenkouSpanA line buffer element index.

Return Value

The buffer element of the SenkouSpanA line of the specified index, or EMPTY_VALUE if there is no correct data.

SenkouSpanB

Returns the buffer element of the SenkouSpanB line by the specified index.

```
double  SenkouSpanB(
   int  index       // index
   )
```

Parameters

index

[in] SenkouSpanB line buffer element index.

Return Value

The buffer element of the SenkouSpanB line of the specified index, or EMPTY_VALUE if there is no correct data.

ChinkouSpan

Returns the buffer element of the ChinkouSpan line by the specified index.

```
double  ChinkouSpan(
   int  index       // index
   )
```

Parameters

index

[in] ChinkouSpan line buffer element index.

Return Value

The buffer element of the ChinkouSpan line of the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ICHIMOKU for CiIchimoku).

CiMA

CiMA is a class intended for using the Moving Average technical indicator.

Description

CiMA class provides the creation, setup, and access to the data of the Moving Average indicator.

Declaration

```
class CiMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method (value of ENUM_MA_METHOD enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             string,        // symbol
   ENUM_TIMEFRAMES    period,        // period
   int                ma_period,     // averaging period
   int                ma_shift,      // shift
   ENUM_MA_METHOD     ma_method,     // averaging method
   int                applied        // price type, handle
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_MA for CiMA).

CiSAR

CiSAR is a class intended for using the Parabolic Stop And Reverse System technical indicator.

Description

CiSAR class provides the creation, setup, and access to the data of the Parabolic Stop And Reverse System indicator.

Declaration

```
class CiSAR: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiSAR

Class Methods by Groups

Attributes	
SarStep	Returns the step of price increment
Maximum	Returns the maximum value of the step
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

SarStep

Returns the step of price increment.

```
double  SarStep() const
```

Return Value

The step of price increment, defined at the indicator creation.

Maximum

Returns the maximum value of the step.

```
double  Maximum() const
```

Return Value

The maximum value of the step, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,       // symbol
   ENUM_TIMEFRAMES   period,       // period
   double            step,         // step
   double            maximum       // coefficient
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

step

[in] Step for the velocity increasing.

maximum

[in] Price following coefficient.

Return Value

true - successful, false - cannot change the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_SAR for CiSAR).

CiStdDev

CiStdDev is a class intended for using the Standard Deviation technical indicator.

Description

CiStdDev class provides the creation, setup, and access to the data of the Standard Deviation indicator.

Declaration

```
   class CiStdDev: public CIndicator
```

Title

```
   #include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiStdDev

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method (value of ENUM_MA_METHOD enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,         // symbol
   ENUM_TIMEFRAMES    period,         // period
   int                ma_period,      // averaging period
   int                ma_shift,       // shift
   ENUM_MA_METHOD     ma_method,      // averaging method
   int                applied         // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index if successful, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_STDDEV for CiStdDev).

CiDEMA

CiDEMA is a class intended for using the Double Exponential Moving Average technical indicator.

Description

CiDEMA class provides the creation, setup, and access to the data of the Double Exponential Moving Average indicator.

Declaration

```
   class CiDEMA: public CIndicator
```

Title

```
   #include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiDEMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           string,         // symbol
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period,      // averaging period
   int              ind_shift,      // shift
   int              applied         // price type, handle
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_DEMA for CiDEMA).

CiTEMA

CiTEMA is a class intended for using the Triple Exponential Moving Average technical indicator.

Description

CiTEMA class provides the creation, setup, and access to the data of the Triple Exponential Moving Average indicator.

Declaration

```
class CiTEMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiTEMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,         // symbol
   ENUM_TIMEFRAMES   period,         // period
   int               ma_period,      // averaging period
   int               ma_shift,       // shift
   int               applied         // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index        // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_TEMA for CiTEMA).

CiFrAMA

CiFrAMA is a class intended for using the Fractal Adaptive Moving Average technical indicator.

Description

CiFrAMA class provides the creation, setup, and access to the data of the Fractal Adaptive Moving Average indicator.

Declaration

```
class CiFrAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiFrAMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,        // symbol
   ENUM_TIMEFRAMES  period,        // period
   int              ma_period,     // averaging period
   int              ma_shift,      // shift
   int              applied        // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index if successful, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_FRAMA for CiFrAMA).

CiAMA

CiAMA is a class intended for using the Adaptive Moving Average technical indicator.

Description

CiAMA class provides the creation, setup, and access to the data of the Adaptive Moving Average indicator.

Declaration

```
class CiAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiAMA

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
FastEmaPeriod	Returns the averaging period for the fast EMA
SlowEmaPeriod	Returns the averaging period for the slow EMA
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            string,              // symbol
   ENUM_TIMEFRAMES   period,              // period
   int               ma_period,           // averaging period
   int               fast_ema_period,     // fast EMA period
   int               slow_ema_period,     // slow EMA period
   int               ind_shift,           // shift
   int               applied              // price type, handle
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_AMA for CiAMA).

CiVIDyA

CiVIDyA is a class intended for using the Variable Index Dynamic Average technical indicator.

Description

CiVIDyA class provides the creation, setup, and access to the data of the Variable Index Dynamic Average indicator.

Declaration

```
class CiVIDyA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiVIDyA

Class Methods by Groups

Attributes	
CmoPeriod	Returns the period for Momentum
EmaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

CmoPeriod

Closes the chart bound to the class instance.

```
int  CmoPeriod() const
```

EmaPeriod

Returns the chart symbol.

```
int  EmaPeriod() const
```

Return Value

Returns a chart symbol bound to a class instance. "" - no bound chart.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Return Value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,          // symbol
   ENUM_TIMEFRAMES   period,          // period
   int               cmo_period,      // momentum period
   int               ema_period,      // averaging period
   int               ind_shift,       // shift
   int               applied          // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

cmo_period

[in] Momentum period.

ema_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_VIDYA for CiVIDyA).

Oscillator Classes

This group of chapters contains the technical details of Oscillators classes, as well as descriptions of the appropriate components of the MQL5 Standard Library.

Class/group	Description
CiATR	Average True Range
CiBearsPower	Bears Power
CiBullsPower	Bulls Power
CiCCI	Commodity Channel Index
CiChaikin	Chaikin Oscillator
CiDeMarker	DeMarker
CiForce	Force Index
CiMACD	Moving Averages Convergence-Divergence
CiMomentum	Momentum
CiOsMA	Moving Average of Oscillator (MACD histogram)
CiRSI	Relative Strength Index
CiRVI	Relative Vigor Index
CiStochastic	Stochastic Oscillator
CiWPR	Williams' Percent Range
CiTriX	Triple Exponential Moving Averages Oscillator

CiATR

CiATR is a class intended for using the Average True Range technical indicator.

Description

CiATR class provides the creation, setup, and access to the data of the Average True Range indicator.

Declaration

```
class CiATR: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiATR

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,         // symbol
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period       // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ATR for CiATR).

CiBearsPower

CiBearsPower is a class intended for using the Bears Power technical indicator.

Description

CiBearsPower class provides the creation, setup, and access to the data of the Bears Power indicator.

Declaration

```
class CiBearsPower: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiBearsPower

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,         // symbol
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period       // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_BEARS for CiBearsPower).

CiBullsPower

CiBullsPower is a class intended for using the Bulls Power technical indicator.

Description

CiBullsPower class provides the creation, setup, and access to the data of the Bulls Power indicator.

Declaration

```
class CiBullsPower: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiBullsPower

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,         // symbol
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period       // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_BULLS for CiBullsPower).

CiCCI

CiCCI is a class intended for using the Commodity Channel Index technical indicator.

Description

CiCCI class provides the creation, setup, and access to the data of the Commodity Channel Index indicator.

Declaration

```
   class CiCCI: public CIndicator
```

Title

```
   #include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiCCI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,         // symbol
   ENUM_TIMEFRAMES    period,         // period
   int                ma_period,      // averaging period
   int                applied         // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_CCI for CiCCI).

CiChaikin

CiChaikin is a class intended for using the Chaikin Oscillator technical indicator.

Description

CiChaikin class provides the creation, setup, and access to the data of the Chaikin Oscillator indicator.

Declaration

```
class CiChaikin: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiChaikin

Class Methods by Groups

Attributes	
FastMaPeriod	Returns the averaging period for the fast MA
SlowMaPeriod	Returns the averaging period for the slow MA
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

FastMaPeriod

Returns the averaging period for the fast EMA.

```
int  FastMaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowMaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowMaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation (ENUM_MA_METHOD enumeration value).

Applied

Returns the object (volume type) to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Object (volume type) to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                 symbol,              // symbol
   ENUM_TIMEFRAMES        period,              // period
   int                    fast_ma_period,      // fast EMA period
   int                    slow_ma_period,      // slow EMA period
   ENUM_MA_METHOD         ma_method,           // averaging method
   ENUM_APPLIED_VOLUME    applied              // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

fast_ma_period

[in] Period for fast EMA.

slow_ma_period

[in] Period for slow EMA.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Object (volume type) to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_CHAIKIN for CiChaikin).

CiDeMarker

CiDeMarker is a class intended for using the DeMarker technical indicator.

Description

CiDeMarker class provides the creation, setup, and access to the data of the DeMarker indicator.

Declaration

```
class CiDeMarker: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiDeMarker

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,         // symbol
   ENUM_TIMEFRAMES    period,         // period
   int                ma_period       // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_DEMARKER for CiDeMarker).

CiForce

CiForce is a class intended for using the Force Index technical indicator.

Description

CiForce class provides the creation, setup, and access to the data of the Force Index indicator.

Declaration

```
class CiForce: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiForce

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
MaMethod	Returns the averaging method
Applied	Returns the object (volume type) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the object (volume type) to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Object (volume type) to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                symbol,         // symbol
   ENUM_TIMEFRAMES       period,         // period
   int                   ma_period,      // averaging period
   ENUM_MA_METHOD        ma_method,      // averaging method
   ENUM_APPLIED_VOLUME   applied         // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Object (volume type) to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_FORCE for CiForce).

CiMACD

CiMACD is a class intended for using the Moving Averages Convergence-Divergence technical indicator.

Description

CiMACD class provides the creation, setup, and access to the data of the Moving Averages Convergence-Divergence indicator.

Declaration

```
class CiMACD: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiMACD

Class Methods by Groups

Attributes	
FastEmaPeriod	Returns the averaging period of the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int  SignalPeriod() const
```

Return Value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,              // symbol
   ENUM_TIMEFRAMES    period,              // period
   int                fast_ema_period,     // fast EMA period
   int                slow_ema_period,     // slow EMA period
   int                signal_period,       // signal period
   int                applied              // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index        // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_MACD for CiMACD).

CiMomentum

CiMomentum is a class intended for using the Momentum technical indicator.

Description

CiMomentum class provides the creation, setup, and access to the data of the Momentum indicator.

Declaration

```
   class CiMomentum: public CIndicator
```

Title

```
   #include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiMomentum

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the object (volume type) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,         // symbol
   ENUM_TIMEFRAMES  period,         // period
   int              ma_period,      // averaging period
   int              applied         // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_MOMENTUM for CiMomentum).

CiOsMA

CiOsMA is a class intended for using the Moving Average of Oscillator (MACD histogram) technical indicator.

Description

CiOsMA class provides the creation, setup, and access to the data of the Moving Average of Oscillator (MACD histogram) indicator.

Declaration

```
class CiOsMA: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiOsMA

Class Methods by Groups

Attributes	
FastEmaPeriod	Returns the averaging period of the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Return Value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Return Value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int  SignalPeriod() const
```

Return Value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,              // symbol
   ENUM_TIMEFRAMES   period,              // period
   int               fast_ema_period,     // fast EMA period
   int               slow_ema_period,     // slow EMA period
   int               signal_period,       // signal line period
   int               applied              // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_OSMA for CiOsMA).

CiRSI

CiRSI is a class intended for using the Relative Strength Index technical indicator.

Description

CiRSI class provides the creation, setup, and access to the data of the Relative Strength Index indicator.

Declaration

```
   class CiRSI: public CIndicator
```

Title

```
   #include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiRSI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,        // symbol
   ENUM_TIMEFRAMES  period,        // period
   int              ma_period,     // averaging period
   int              applied        // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int   index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_RSI for CiRSI).

CiRVI

CiRVI is a class intended for using the Relative Vigor Index technical indicator.

Description

CiRVI class provides the creation, setup, and access to the data of the Relative Vigor Index indicator.

Declaration

```
class CiRVI: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiRVI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,          // symbol
   ENUM_TIMEFRAMES    period,          // period
   int                ma_period        // averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_RVI for CiRVI).

CiStochastic

CiStochastic is a class intended for using the Stochastic Oscillator technical indicator.

Description

CiStochastic class provides the creation, setup, and access to the data of the Stochastic Oscillator indicator.

Declaration

```
   class CiStochastic: public CIndicator
```

Title

```
   #include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiStochastic

Class Methods by Groups

Attributes	
Kperiod	Returns the averaging period for the %K line
Dperiod	Returns the averaging period for the %D line
Slowing	Returns the slowing period
MaMethod	Returns the averaging method
PriceField	Price type (Low/High or Close/Close) to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data of the main line
Signal	Returns the buffer data of the signal line
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Kperiod

Returns the averaging period for the %K line.

```
int  Kperiod() const
```

Return Value

Returns the averaging period for the %K line, defined at the indicator creation.

Dperiod

Returns the averaging period for the %D line.

```
int  Dperiod() const
```

Return Value

Returns the averaging period for the %D line, defined at the indicator creation.

Slowing

Returns the period of slowing.

```
int  Slowing() const
```

Return Value

Returns the period of slowing, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation (ENUM_MA_METHOD enumeration value).

PriceField

Returns the object (Low/High or Close/Close) to apply.

```
ENUM_STO_PRICE  PriceField() const
```

Return Value

The object (Low/High or Close/Close) to apply, defined at the indicator creation (ENUM_STO_PRICE enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string              symbol,           // symbol
   ENUM_TIMEFRAMES     period,           // period
   int                 Kperiod,          // %K period
   int                 Dperiod,          // %D period
   int                 slowing,          // slowing period
   ENUM_MA_METHOD      ma_method,        // averaging method
   ENUM_STO_PRICE      price_field       // application
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

Kperiod

[in] Averaging period of %K indicator.

Dperiod

[in] Averaging period of %D indicator.

slowing

[in] Slowing period.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

price_field

[in] Object (Low/High or Close/Close) to apply (ENUM_STO_PRICE enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the main line buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Main line buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int   index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

The buffer element of the signal line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_STOCHASTIC for CiStochastic).

CiTriX

CiTriX is a class intended for using the Triple Exponential Moving Averages Oscillator technical indicator.

Description

CiTriX class provides the creation, setup, and access to the data of the Triple Exponential Moving Averages Oscillator indicator.

Declaration

```
   class CiTriX: public CIndicator
```

Title

```
   #include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiTriX

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,          // symbol
   ENUM_TIMEFRAMES   period,          // period
   int               ma_period,       // averaging period
   int               applied          // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

applied

[in] Price type of handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_TRIX for CiTriX).

CiWPR

CiWPR is a class intended for using the Williams' Percent Range technical indicator.

Description

CiWPR class provides the creation, setup, and access to the data of the Williams' Percent Range indicator.

Declaration

```
class CiWPR: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiWPR

Class Methods by Groups

Attributes	
CalcPeriod	Returns the calculation period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

CalcPeriod

Returns the period for calculation.

```
int  CalcPeriod() const
```

Return Value

Returns the period for calculation, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,           // symbol
   ENUM_TIMEFRAMES    period,           // period
   int                calc_period       // calculation period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

calc_period

[in] Period for calculation.

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_WPR for CiWPR).

Volume Indicators

This group of chapters contains technical details of Volume indicator classes and descriptions of all the appropriate key components of the MQL5 Standard Library.

Class/group	Description
CiAD	Accumulation/Distribution
CiMFI	Money Flow Index
CiOBV	On Balance Volume
CiVolumes	Volumes

CiAD

CiAD is a class intended for using the Accumulation/Distribution technical indicator.

Description

CiAD class provides the creation, setup, and access to the data of the Accumulation/Distribution indicator.

Declaration

```
class CiAD: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiAD

Class Methods by Groups

Attributes	
Applied	Returns the calculation period
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                 symbol,       // symbol
   ENUM_TIMEFRAMES        period,       // period
   ENUM_APPLIED_VOLUME    applied       // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_AD for CiAD).

CiMFI

CiMFI is a class intended for using the Money Flow Index technical indicator.

Description

CiMFI class provides the creation, setup, and access to the data of the Money Flow Index indicator.

Declaration

```
   class CiMFI: public CIndicator
```

Title

```
   #include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiMFI

Class Methods by Groups

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the volume type to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Return Value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                symbol,         // symbol
   ENUM_TIMEFRAMES       period,         // period
   int                   ma_period,      // averaging period
   ENUM_APPLIED_VOLUME   applied         // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

ma_period

[in] Averaging period.

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_MFI for CiMFI).

CiOBV

CiOBV is a class intended for using the On Balance Volume technical indicator.

Description

CiOBV class provides the creation, setup, and access to the data of the On Balance Volume indicator.

Declaration

```
class CiOBV: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiOBV

Class Methods by Groups

Attributes	
Applied	Returns the volume type to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                symbol,       // symbol
   ENUM_TIMEFRAMES       period,       // period
   ENUM_APPLIED_VOLUME   applied       // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_OBV for CiOBV).

CiVolumes

CiVolumes is a class intended for using the Volumes technical indicator.

Description

CiVolumes class provides the creation, setup, and access to the data of the Volumes indicator.

Declaration

```
class CiVolumes: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiVolumes

Class Methods by Groups

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                symbol,       // symbol
   ENUM_TIMEFRAMES       period,       // period
   ENUM_APPLIED_VOLUME   applied       // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_VOLUMES for CiVolumes).

Bill Williams Indicators

This group of chapters contains technical details of Bill Williams indicator classes and descriptions of all the appropriate components of the MQL5 Standard Library.

Class/group	Description
CiAC	Accelerator Oscillator
CiAlligator	Alligator
CiAO	Awesome Oscillator
CiFractals	Fractals
CiGator	Gator Oscillator
CiBWMFI	Market Facilitation Index

CiAC

CiAC is a class intended for using the Accelerator Oscillator technical indicator.

Description

CiAC class provides the creation, setup, and access to the data of the Accelerator Oscillator indicator.

Declaration

```
class CiAC: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiAC

Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int   index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_AC for CiAC).

CiAlligator

CiAlligator is a class intended for using the Alligator technical indicator.

Description

CiAlligator class provides the creation, setup, and access to the data of the Alligator indicator.

Declaration

```
class CiAlligator: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiAlligator

Class Methods by Groups

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeth line
TeethShift	Returns the horizontal shift of the Teeth line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create	
Create	Creates the indicator
Data Access	
Jaw	Returns the buffer data of the Jaws line buffer
Teeth	Returns the buffer data of the Teeth line buffer

Attributes	
Lips	Returns the buffer data of the Lips line buffer
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

JawPeriod

Returns the averaging period for the Jaw line.

```
int  JawPeriod() const
```

Return Value

Returns the averaging period for the Jaw line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift() const
```

Return Value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int  TeethPeriod() const
```

Return Value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeth line.

```
int  TeethShift() const
```

Return Value

Horizontal shift of the Teeth line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int  LipsPeriod() const
```

Return Value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int  LipsShift() const
```

Return Value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,            // symbol
   ENUM_TIMEFRAMES    period,            // period
   int                jaw_period,        // jaws period
   int                jaw_shift,         // jaws shift
   int                teeth_period,      // teeth period
   int                teeth_shift,       // teeth shift
   int                lips_period,       // lips period
   int                lips_shift,        // lips shift
   ENUM_MA_METHOD     ma_method,         // averaging method
   int                applied            // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

jaw_period

[in] Jaws averaging period.

jaw_shift

[in] Jaws horizontal shift.

teeth_period

[in] Teeth averaging period.

teeth_shift

[in] Teeth horizontal shift.

lips_period

[in] Lips averaging period.

lips_shift

[in] Lips horizontal shift.

ma_method

[in] Moving average method (ENUM_MA_METHOD enumeration value).

applied

[in] Price type, handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Jaw

Returns the buffer element of the Jaws line by the specified index.

```
double  Jaw(
   int  index      // index
   )
```

Parameters

index

[in] Jaws line buffer element index.

Return Value

The buffer element of the Jaws line by the specified index, or EMPTY_VALUE if there is no correct data.

Teeth

Returns the buffer element of the Teeth line by the specified index.

```
double  Teeth(
   int  index       // index
   )
```

Parameters

index

[in] Teeth line buffer element index.

Return Value

The buffer element of the Teeth line by the specified index, or EMPTY_VALUE if there is no correct data.

Lips

Returns the buffer element of the Lips line by the specified index.

```
double  Lips(
   int  index       // index
   )
```

Parameters

index

[in] Lips line buffer element index.

Return Value

The buffer element of the Lips line by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_ALLIGATOR for CiAlligator).

CiAO

CiAO is a class intended for using the Awesome Oscillator technical indicator.

Description

CiAO class provides the creation, setup, and access to the data of the Awesome Oscillator indicator.

Declaration

```
class CiAO: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiAO

Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer element
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,       // symbol
   ENUM_TIMEFRAMES   period        // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index       // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element of the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_AO for CiAO).

CiFractals

CiFractals is a class intended for using the Fractals technical indicator.

Description

CiFractals class provides the creation, setup, and access to the data of the Fractals indicator.

Declaration

```
class CiFractals: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiFractals

Class Methods by Groups

Create	
Create	Creates the indicator
Data Access	
Upper	Returns the data of the upper buffer
Lower	Returns the data of the lower buffer
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear,

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string            symbol,      // symbol
   ENUM_TIMEFRAMES   period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the element of the upper buffer by the specified index.

```
double  Upper(
   int  index       // index
   )
```

Parameters

index

[in] Upper buffer element index.

Return Value

The element of the upper buffer by the specified index, or EMPTY_VALUE if there is no correct data.

Lower

Returns the element of the lower buffer by the specified index.

```
double  Lower(
   int  index      // index
   )
```

Parameters

index

[in] Lower buffer element index.

Return Value

The element of the lower buffer by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_FRACTALS for CiFractals).

CiGator

CiGator is a class intended for using the Gator Oscillator technical indicator.

Description

CiGator class provides the creation, setup, and access to the data of the Gator Oscillator indicator.

Declaration

```
   class CiGator: public CIndicator
```

Title

```
   #include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiGator

Class Methods by Groups

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeth line
TeethShift	Returns the horizontal shift of the Teeth line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Upper	Returns the data of the upper buffer
Lower	Returns the data of the lower buffer

Attributes	
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

JawPeriod

Returns the averaging period for the Jaws line.

```
int  JawPeriod() const
```

Return Value

Returns the averaging period for the Jaws line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift() const
```

Return Value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int  TeethPeriod() const
```

Return Value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeth line.

```
int  TeethShift() const
```

Return Value

Horizontal shift of the Teeth line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int  LipsPeriod() const
```

Return Value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int  LipsShift() const
```

Return Value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Return Value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Return Value

Price type or handle to apply, defined at the indicator creation.

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string             symbol,            // symbol
   ENUM_TIMEFRAMES    period,            // period
   int                jaw_period,        // jaws period
   int                jaw_shift,         // jaws shift
   int                teeth_period,      // teeth period
   int                teeth_shift,       // teeth shift
   int                lips_period,       // lips period
   int                lips_shift,        // lips shift
   ENUM_MA_METHOD     ma_method,         // averaging method
   int                applied            // price type, handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

jaw_period

[in] Jaws averaging period.

jaw_shift

[in] Jaws horizontal shift.

teeth_period

[in] Teeth averaging period.

teeth_shift

[in] Teeth horizontal shift.

lips_period

[in] Lips averaging period.

lips_shift

[in] Lips horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration value).

applied

[in] Price type or handle to apply.

Return Value

true - successful, false - cannot create the indicator.

Upper

Returns the element of the upper buffer by the specified index.

```
double  Upper(
   int  index       // index
   )
```

Parameters

index

[in] Upper buffer element index.

Return Value

The upper buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Lower

Returns the element of the lower buffer by the specified index.

```
double  Upper(
   int  index      // index
   )
```

Parameters

index

[in] Lower buffer element index.

Return Value

The element of the lower buffer by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_GATOR for CiGator).

CiBWMFI

CiBWMFI is a class intended for using the Market Facilitation Index by Bill Williams technical indicator.

Description

CiBWMFI class provides the creation, setup, and access to the data of the Market Facilitation Index by Bill Williams indicator.

Declaration

```
   class CiBWMFI: public CIndicator
```

Title

```
   #include <Indicators\BillWilliams.mqh>
```

Inheritance hierarchy

- CObject
  - CArray
    - CArrayObj
      - CSeries
        - CIndicator
          - CiBWMFI

Class Methods by Groups

Attributes	
Applied	Returns the volume type to apply
Create	
Create	Creates the indicator
Data Access	
Main	Returns the buffer data
Input/output	
virtual Type	Virtual identification method

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

Methods inherited from class CArray

Step, Step, Total, Available, Max, IsSorted, SortMode, Clear, Sort

Methods inherited from class CArrayObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Compare

FreeMode, FreeMode, Save, Load, CreateElement, Reserve, Resize, Shutdown, Add, AddArray, Insert, InsertArray, AssignArray, At, Update, Shift, Detach, Delete, DeleteRange, Clear, CompareArray, InsertSort, Search, SearchGreat, SearchLess, SearchGreatOrEqual, SearchLessOrEqual, SearchFirst, SearchLast

Methods inherited from class CSeries

Name, BuffersTotal, BufferSize, Timeframe, Symbol, Period, PeriodDescription, RefreshCurrent

Methods inherited from class CIndicator

Handle, Status, FullRelease, Redrawer, Create, BufferResize, BarsCalculated, GetData, GetData, GetData, GetData, Minimum, MinValue, Maximum, MaxValue, Refresh, AddToChart, DeleteFromChart, MethodDescription, PriceDescription, VolumeDescription

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Return Value

Volume type to apply, defined at the indicator creation (ENUM_APPLIED_VOLUME enumeration value).

Create

Creates the indicator with specified parameters. Use Refresh() and GetData() to update and get the indicator values.

```
bool  Create(
   string                 symbol,        // symbol
   ENUM_TIMEFRAMES        period,        // period
   ENUM_APPLIED_VOLUME    applied        // volume type
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration value).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration value).

Return Value

true - successful, false - cannot create the indicator.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // index
   )
```

Parameters

index

[in] Buffer element index.

Return Value

Buffer element by the specified index, or EMPTY_VALUE if there is no correct data.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_BWMFI for CiBWMFI).

CiCustom

CiCustom is a class intended for using the custom technical indicators.

Description

CiCustom class provides the creation, setup, and access to the data of a custom indicator.

Declaration

```
class CiCustom: public CIndicator
```

Title

```
#include <Indicators\Custom.mqh>
```

Class Methods by Groups

Attributes	
NumBuffers	Sets the number of buffers
NumParams	Gets the number of parameters used when creating an indicator
ParamType	Gets the type of the specified parameter
ParamLong	Gets the value of the specified parameter of integer type
ParamDouble	Gets the value of the specified parameter of double type
ParamString	Gets the value of the specified parameter of string type
Input/output	
virtual Type	Virtual identification method

NumBuffers

Sets the number of buffers.

```
bool  NumBuffers(
```

Return Value

true - successful, false - cannot set the necessary number of buffers.

NumParams

Gets the number of parameters used when creating an indicator.

```
int  NumParams() const
```

Return Value

Number of parameters, used in creation of the indicator.

ParamType

Gets a type of the parameter.

```
ENUM_DATATYPE  ParamType(
   int  index      // index
   ) const
```

Parameters

index

[in] Parameter index.

Return Value

Returns the data type of the specified parameter, used in indicator creation.

Note

If parameter index is invalid, it returns WRONG_VALUE.

ParamLong

Gets the value of specified parameter of long type.

```
long  ParamLong(
   int   index       // index
   ) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified parameter of long type, used in creation of the indicator.

Note

If the parameter index is invalid or the parameter is not of long type, it returns 0.

ParamDouble

Gets the value of specified parameter of double type.

```
double  ParamDouble(
   int   index       // index
   ) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified parameter of double type, used in creation of the indicator.

Note

If the parameter index is invalid or the parameter type is not of double type, it returns EMPTY_VALUE.

ParamString

Gets the value of specified parameter of string type.

```
string  ParamString(
   int  index      // index
   ) const
```

Parameters

index

[in] Parameter index.

Return Value

The value of specified string parameter, used in creation of the indicator.

Note

If the number is invalid or the parameter is not of string type, it returns an empty string.

Type

Virtual identification method.

```
virtual int  Type() const
```

Return Value

Indicator type (IND_CUSTOM for CiCustom).

Trade Classes

This section contains technical details of working with trade classes and description of the relevant components of the MQL5 standard library.

Using trade classes will save time when creating custom programs (Expert Advisors).

MQL5 Standard Library (in terms of trade classes) is placed in the terminal working directory, in the Include\Trade folder.

Class/Group	Description
CAccountInfo	Class for working with trade account properties
CSymbolInfo	Class for working with trade instrument properties
COrderInfo	Class for working with pending order properties
CHistoryOrderInfo	Class for working with history order properties
CPositionInfo	Class for working with open position properties
CDealInfo	Class for working with history deal properties
CTrade	Class for trade operations execution
CTerminalInfo	Class for getting the properties of the terminal environment

CAccountInfo

CAccountInfo is a class for easy access to the currently opened trade account properties.

Description

CAccountInfo class provides easy access to the currently opened trade account properties.

Declaration

```
class CAccountInfo : public CObject
```

Title

```
#include <Trade\AccountInfo.mqh>
```

Inheritance hierarchy

CObject

CAccountInfo

Class methods by groups

Access to integer type properties	
Login	Gets the account number
TradeMode	Gets the trade mode
TradeModeDescription	Gets the trade mode as a string
Leverage	Gets the amount of given leverage
StopoutMode	Gets the mode of stop out setting
StopoutModeDescription	Gets the mode of stop out setting as a string
TradeAllowed	Gets the flag of trade allowance
TradeExpert	Gets the flag of automated trade allowance
LimitOrders	Gets the maximal number of allowed pending orders
MarginMode	Gets margin calculation mode
MarginModeDescription	Gets margin calculation mode as a string
Access to double type properties	
Balance	Gets the balance of account
Credit	Gets the amount of given credit
Profit	Gets the amount of current profit on account

Access to integer type properties	
Equity	Gets the amount of current equity on account
Margin	Gets the amount of reserved margin
FreeMargin	Gets the amount of free margin
MarginLevel	Gets the level of margin
MarginCall	Gets the level of margin for deposit
MarginStopOut	Gets the level of margin for Stop Out
Access to text properties	
Name	Gets the client name
Server	Gets the trade server name
Currency	Gets the deposit currency name
Company	Gets the company name that serves an account
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Additional methods	
OrderProfitCheck	Gets the evaluated profit based on the parameters passed
MarginCheck	Gets the amount of margin required to execute trade operation
FreeMarginCheck	Gets the amount of free margin left after execution of trade operation
MaxLotCheck	Gets the maximal possible volume of trade operation

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Login

Gets the account number.

```
long  Login() const
```

Return Value

Account number.

TradeMode

Gets the trade mode.

```
ENUM_ACCOUNT_TRADE_MODE  TradeMode() const
```

Return Value

Trade mode from the ENUM_ACCOUNT_TRADE_MODE enumeration.

TradeModeDescription

Gets the trade mode as a string.

```
string  TradeModeDescription() const
```

Return Value

Trade mode as a string.

Leverage

Gets the amount of given leverage.

```
long  Leverage() const
```

Return Value

Amount of given leverage.

StopoutMode

Gets the mode of the Stop Out level specification.

```
ENUM_ACCOUNT_STOPOUT_MODE  StopoutMode() const
```

Return Value

The stop out setting mode from the ENUM_ACCOUNT_STOPOUT_MODE enumeration.

StopoutModeDescription

Gets the mode of the Stop Out level specification as a string.

```
string  StopoutModeDescription() const
```

Return Value

The stop out setting mode as a string.

MarginMode

Gets the margin calculation mode.

```
ENUM_ACCOUNT_MARGIN_MODE  MarginMode() const
```

Return Value

The margin calculation mode from the ENUM_ACCOUNT_MARGIN_MODE enumeration.

MarginModeDescription

Gets the margin calculation mode as a string.

```
string  MarginModeDescription() const
```

Return Value

Margin calculation mode as a string.

TradeAllowed

Gets the flag of trade allowance.

```
bool  TradeAllowed() const
```

Return Value

Flag of trade allowance.

TradeExpert

Gets the flag of automated trade allowance.

```
bool  TradeExpert() const
```

Return Value

Flag of automated trade allowance.

LimitOrders

Gets the maximal number of allowed pending orders

```
int  LimitOrders() const
```

Return Value

The maximal number of allowed pending orders.

Note

0 - no limits.

Balance

Gets the balance of account.

```
double  Balance() const
```

Return Value

The balance of account (in deposit currency).

Credit

Gets the amount of given credit.

```
double  Credit() const
```

Return Value

Amount of given credit (in deposit currency).

Profit

Gets the amount of current profit on account.

```
double  Profit() const
```

Return Value

Amount of current profit on account (in deposit currency).

Equity

Gets the amount of current equity on account.

```
double  Equity() const
```

Return Value

Amount of current equity on account (in deposit currency).

Margin

Gets the amount of reserved margin.

```
double  Margin() const
```

Return Value

Amount of reserved margin (in deposit currency).

FreeMargin

Gets the amount of free margin.

```
double  FreeMargin() const
```

Return Value

Amount of free margin (in deposit currency).

MarginLevel

Gets the level of margin.

```
double  MarginLevel() const
```

Return Value

Level of margin.

MarginCall

Gets the level of margin for a deposit.

```
double  MarginCall() const
```

Return Value

Level of margin for a deposit.

MarginStopOut

Gets the level of margin for Stop Out.

```
double  MarginStopOut() const
```

Return Value

Level of margin for Stop Out.

Name

Gets the client name.

```
string  Name() const
```

Return Value

Client name.

Server

Gets the trade server name.

```
string  Server() const
```

Return Value

Trade server name.

Currency

Gets the deposit currency name.

```
string  Currency() const
```

Return Value

Deposit currency name.

Company

Gets the company name, that serves an account.

```
string  Company() const
```

Return Value

Company name that serves an account.

InfoInteger

Gets the value of specified integer type property.

```
long  InfoInteger(
   ENUM_ACCOUNT_INFO_INTEGER  prop_id     // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_INTEGER enumeration.

Return Value

Value of long type.

InfoDouble

Gets the value of specified double type property.

```
double  InfoDouble(
   ENUM_ACCOUNT_INFO_DOUBLE  prop_id      // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_DOUBLE enumeration.

Return Value

Value of double type.

InfoString

Gets the value of specified string type property.

```
string  InfoString(
   ENUM_ACCOUNT_INFO_STRING  prop_id      // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_STRING enumeration.

Return Value

Value of string type.

OrderProfitCheck

The function calculates the profit for the current account, based on the parameters passed. The function is used for pre-evaluation of the result of a trade operation. The value is returned in the account currency.

```
double  OrderProfitCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,     // order type (ORDER_TYPE_BUY or ORDER_TYI
   double               volume,              // volume
   double               price_open,          // position open price
   double               price_close          // position close price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from ENUM_ORDER_TYPE enumeration.

volume

[in] Volume of trade operation.

price_open

[in] Open price.

price_close

[in] Close price.

Return Value

If successful, it returns amount of profit or EMPTY_VALUE in the case of error.

MarginCheck

Gets the amount of margin, required for trade operation.

```
double  MarginCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,      // order type (ORDER_TYPE_BUY or ORDER_TY
   double               volume,              // volume
   double               price                // price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from ENUM_ORDER_TYPE enumeration.

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Return Value

Amount of margin required for trade operation.

FreeMarginCheck

Gets the amount of free margin left after trade operation.

```
double  FreeMarginCheck(
   const string         symbol,                // symbol
   ENUM_ORDER_TYPE      trade_operation,       // order type (ORDER_TYPE_BUY or ORDER_TYI
   double               volume,                // volume
   double               price                  // price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from ENUM_ORDER_TYPE enumeration.

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Return Value

Amount of free margin left after trade operation.

MaxLotCheck

Gets the maximum possible volume of trade operation.

```
double  MaxLotCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,     // order type (ORDER_TYPE_BUY or ORDER_TYI
   double               price,               // price
   double               percent=100          // percent of available margin (default is
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation from ENUM_ORDER_TYPE enumeration.

price

[in] Price of trade operation.

percent=100

[in] Percent of available margin (in %) to be used for trade operation.

Return Value

Maximum possible volume of trade operation.

CSymbolInfo

CSymbolInfo is a class for easy access to the symbol properties.

Description

CSymbolInfo class provides access to the symbol properties.

Declaration

```
class CSymbolInfo : public CObject
```

Title

```
#include <Trade\SymbolInfo.mqh>
```

Inheritance hierarchy

CObject

CSymbolInfo

Class methods by groups

Controlling	
Refresh	Refreshes the symbol data
RefreshRates	Refreshes the symbol quotes
Properties	
Name	Gets/sets symbol name
Select	Gets/sets the "Market Watch" symbol flag
IsSynchronized	Checks the symbol synchronization with server
Volumes	
Volume	Gets the volume of last deal
VolumeHigh	Gets the maximal volume for a day
VolumeLow	Gets the minimal volume for a day
Miscellaneous	
Time	Gets the time of last quote
Spread	Gets the amount of spread (in points)
SpreadFloat	Gets the flag of floating spread
TicksBookDepth	Gets the depth of ticks saving
Levels	

Controlling	
StopsLevel	Gets the minimal indent for orders (in points)
FreezeLevel	Gets the distance of freezing trade operations (in points)
Bid prices	
Bid	Gets the current Bid price
BidHigh	Gets the maximal Bid price for a day
BidLow	Gets the minimal Bid price for a day
Ask prices	
Ask	Gets the current Ask price
AskHigh	Gets the maximal Ask price for a day
AskLow	Gets the minimal Ask price for a day
Prices	
Last	Gets the current Last price
LastHigh	Gets the maximal Last price for a day
LastLow	Gets the minimal Last price for a day
Trade modes	
TradeCalcMode	Gets the mode of contract cost calculation
TradeCalcModeDescription	Gets the mode of contract cost calculation as a string
TradeMode	Gets the type of order execution
TradeModeDescription	Gets the type of order execution as a string
TradeExecution	Gets the trade execution mode
TradeExecutionDescription	Gets the execution mode as a string
Swaps	
SwapMode	Gets the swap calculation mode
SwapModeDescription	Gets the swap calculation mode as a string
SwapRollover3days	Gets the day of triple swap charge
SwapRollover3daysDescription	Gets the day of triple swap charge as a string
Margins and flags	
MarginInitial	Gets the value of initial margin
MarginMaintenance	Gets the value of maintenance margin
MarginLong	Gets the rate of margin charging for long positions

Controlling	
MarginShort	Gets the rate of margin charging for short positions
MarginLimit	Gets the rate of margin charging for Limit orders
MarginStop	Gets the rate of margin charging for Stop orders
MarginStopLimit	Gets the rate of margin charging for StopLimit orders
TradeTimeFlags	Gets the flags of allowed expiration modes
TradeFillFlags	Gets the flags of allowed filling modes
Quantization	
Digits	Gets the number of digits after period
Point	Gets the value of one point
TickValue	Gets the tick value (minimal change of price)
TickValueProfit	Gets the calculated tick price for a profitable position
TickValueLoss	Gets the calculated tick price for a losing position
TickSize	Gets the minimal change of price
Contracts sizes	
ContractSize	Gets the amount of trade contract
LotsMin	Gets the minimal volume to close a deal
LotsMax	Gets the maximal volume to close a deal
LotsStep	Gets the minimal step of volume change to close a deal
LotsLimit	Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol
Swaps sizes	
SwapLong	Gets the value of long position swap
SwapShort	Gets the value of short position swap
Text properties	
CurrencyBase	Gets the name of symbol base currency
CurrencyProfit	Gets the profit currency name
CurrencyMargin	Gets the margin currency name
Bank	Gets the name of current quote source
Description	Gets the string description of symbol
Path	Gets the path in symbols tree
Symbol properties	

Controlling	
SessionDeals	Gets the number of deals in the current session
SessionBuyOrders	Gets the number of Buy orders at the moment
SessionSellOrders	Gets the number of Sell orders at the moment
SessionTurnover	Gets the summary of turnover of the current session
SessionInterest	Gets the summary of open interest of the current session
SessionBuyOrdersVolume	Gets the current volume of Buy orders
SessionSellOrdersVolume	Gets the current volume of Sell orders
SessionOpen	Gets the open price of the current session
SessionClose	Gets the close price of the current session
SessionAW	Gets the average weighted price of the current session
SessionPriceSettlement	Gets the settlement price of the current session
SessionPriceLimitMin	Gets the minimal price of the current session
SessionPriceLimitMax	Gets the maximal price of the current session
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Service functions	
NormalizePrice	Returns the value of price, normalized using the symbol properties

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Refresh

Refreshes the symbol data.

```
void  Refresh()
```

Return Value

None.

Note

The symbol should be selected by Name method.

RefreshRates

Refreshes the symbol quotes data.

```
bool  RefreshRates()
```

Return Value

true - success, false - unable to refresh quotes.

Note

The symbol should be selected by Name method.

Name

Gets symbol name.

```
string  Name() const
```

Return Value

Symbol name.

Name

Sets symbol name.

```
bool  Name(string name)
```

Return Value

None.

Select

Gets the "Market Watch" symbol flag.

```
bool  Select() const
```

Return Value

"Market Watch" symbol flag.

Select

Sets the "Market Watch" symbol flag.

```
bool  Select()
```

Return Value

true - success, false - unable to change flag.

Note

The symbol should be selected by Name method.

IsSynchronized

Checks the symbol synchronization with server.

```
bool  IsSynchronized() const
```

Return Value

true - if the symbol is synchronized with server, false - if not.

Note

The symbol should be selected by Name method.

Volume

Gets the volume of last deal.

```
long  Volume() const
```

Return Value

Volume of last deal.

Note

The symbol should be selected by Name method.

VolumeHigh

Gets the maximal volume of the day.

```
long  VolumeHigh() const
```

Return Value

Maximal volume of the day.

Note

The symbol should be selected by Name method.

VolumeLow

Gets the minimal volume of the day.

```
long  VolumeLow() const
```

Return Value

Minimal volume of the day.

Note

The symbol should be selected by Name method.

Time

Gets the time of last quote.

```
datetime  Time() const
```

Return Value

Time of last quote.

Note

The symbol should be selected by Name method.

Spread

Gets the amount of spread (in points).

```
int  Spread() const
```

Return Value

Gets the amount of spread (in points).

Note

The symbol should be selected by Name method.

SpreadFloat

Gets the flag of floating spread.

```
bool  SpreadFloat() const
```

Return Value

Flag of floating spread.

Note

The symbol should be selected by Name method.

TicksBookDepth

Gets the depth of ticks saving.

```
int  TicksBookDepth() const
```

Return Value

Depth of ticks saving.

Note

The symbol should be selected by Name method.

StopsLevel

Gets the minimal stop level for orders (in points).

```
int  StopsLevel() const
```

Return Value

Minimal stop level for orders (in points).

Note

The symbol should be selected by Name method.

FreezeLevel

Gets the freeze level (in points).

```
int  FreezeLevel() const
```

Return Value

Distance of freeze level (in points).

Note

The symbol should be selected by Name method.

Bid

Gets the current Bid price.

```
double  Bid() const
```

Return Value

Current Bid price.

Note

The symbol should be selected by Name method.

BidHigh

Gets the maximal Bid price of the day.

```
double  BidHigh() const
```

Return Value

Maximal Bid price of the day.

Note

The symbol should be selected by Name method.

BidLow

Gets the minimal Bid price of the day.

```
double  BidLow() const
```

Return Value

Minimal Bid price of the day.

Note

The symbol should be selected by Name method.

Ask

Gets the current Ask price.

```
double  Ask() const
```

Return Value

Current Ask price.

Note

The symbol should be selected by Name method.

AskHigh

Gets the maximal Ask price for a day.

```
double  AskHigh() const
```

Return Value

Maximal Ask price of the day.

Note

The symbol should be selected by Name method.

AskLow

Gets the minimal Ask price for a day.

```
double  AskLow() const
```

Return Value

Minimal Ask price of the day.

Note

The symbol should be selected by Name method.

Last

Gets the current Last price.

```
double  Last() const
```

Return Value

Current Last price.

LastHigh

Gets the maximal Last price of the day.

```
double  LastHigh() const
```

Return Value

Maximal Last price of the day.

Note

The symbol should be selected by Name method.

LastLow

Gets the minimal Last price of the day.

```
double  LastLow() const
```

Return Value

Minimal Last price of the day.

Note

The symbol should be selected by Name method.

TradeCalcMode

Gets the mode of contract cost calculation.

```
ENUM_SYMBOL_CALC_MODE  TradeCalcMode() const
```

Return Value

Mode of contract cost calculation from ENUM_SYMBOL_CALC_MODE enumeration.

Note

The symbol should be selected by Name method.

TradeCalcModeDescription

Gets the mode of contract cost calculation as a string.

```
string  TradeCalcModeDescription() const
```

Return Value

Mode of contract cost calculation as a string.

Note

The symbol should be selected by Name method.

TradeMode

Gets the order execution type.

```
ENUM_SYMBOL_TRADE_MODE  TradeMode() const
```

Return Value

Order execution type from ENUM_SYMBOL_TRADE_MODE enumeration.

Note

The symbol should be selected by Name method.

TradeModeDescription

Gets the trade mode as a string.

```
string  TradeModeDescription() const
```

Return Value

Trade mode as a string.

Note

The symbol should be selected by Name method.

TradeExecution

Gets the trade execution mode.

```
ENUM_SYMBOL_TRADE_EXECUTION  TradeExecution() const
```

Return Value

Trade execution mode from ENUM_SYMBOL_TRADE_EXECUTION enumeration.

Note

The symbol should be selected by Name method.

TradeExecutionDescription

Gets the description of trade execution mode as a string.

```
string  TradeExecutionDescription() const
```

Return Value

Trade execution mode as a string.

Note

The symbol should be selected by Name method.

SwapMode

Gets the swap calculation mode.

```
ENUM_SYMBOL_SWAP_MODE  SwapMode() const
```

Return Value

Swap calculation mode from ENUM_SYMBOL_SWAP_MODE enumeration.

Note

The symbol should be selected by Name method.

SwapModeDescription

Gets the swap mode description as a string.

```
string  SwapModeDescription() const
```

Return Value

Swap mode descriprion as a string.

Note

The symbol should be selected by Name method.

SwapRollover3days

Gets the swap rollover day.

```
ENUM_DAY_OF_WEEK  SwapRollover3days() const
```

Return Value

Swap rollover day from ENUM_DAY_OF_WEEK enumeration.

Note

The symbol should be selected by Name method.

SwapRollover3daysDescription

Gets the swap rollover day as a string.

```
string  SwapRollover3daysDescription() const
```

Return Value

Swap rollover day as a string.

Note

The symbol should be selected by Name method.

MarginInitial

Gets the value of initial margin.

```
double  MarginInitial()
```

Return Value

Value of initial margin.

Note

It returns the amount of margin (in margin currency of instrument) that is charged from one lot. Used to check client's equity, when they enter the market.

The symbol should be selected by Name method.

MarginMaintenance

Gets the value of maintenance margin.

```
double  MarginMaintenance()
```

Return Value

Value of maintenance margin.

Note

It returns the amount of margin (in margin currency of instrument) that is charged from one lot. Used to check client's equity, when the account state is changed. If the maintenance margin is equal to 0, then the initial margin is used.

The symbol should be selected by Name method.

MarginLong

Gets the rate of margin charging on long positons.

```
double  MarginLong() const
```

Return Value

Rate of margin charging on long positons.

Note

The symbol should be selected by Name method.

MarginShort

Gets the rate of margin charging on short positons.

```
double  MarginShort() const
```

Return Value

Rate of margin charging on short positons.

Note

The symbol should be selected by Name method.

MarginLimit

Gets the rate of margin charging on Limit orders.

```
double  MarginLimit() const
```

Return Value

Rate of margin charging on Limit orders.

Note

The symbol should be selected by Name method.

MarginStop

Gets the rate of margin charging on Stop orders.

```
double  MarginStop() const
```

Return Value

Rate of margin charging on Stop orders.

Note

The symbol should be selected by Name method.

MarginStopLimit

Gets the rate of margin charging on Stop Limit orders.

```
double  MarginStopLimit() const
```

Return Value

Rate of margin charging on Stop Limit orders.

Note

The symbol should be selected by Name method.

TradeTimeFlags

Gets the flags of allowed expiration modes.

```
int  TradeTimeFlags() const
```

Return Value

Flags of allowed expiration modes.

Note

The symbol should be selected by Name method.

TradeFillFlags

Gets the flags of allowed filling modes.

```
int  TradeFillFlags() const
```

Return Value

Flags of allowed filling modes.

Note

The symbol should be selected by Name method.

Digits

Gets the number of digits after period.

```
int  Digits() const
```

Return Value

The number of digits after period.

Note

The symbol should be selected by Name method.

Point

Gets the value of one point.

```
double  Point() const
```

Return Value

Value of one point.

Note

The symbol should be selected by Name method.

TickValue

Gets the tick value (minimal change of price).

```
double  TickValue() const
```

Return Value

Tick value (minimal change of price).

Note

The symbol should be selected by Name method.

TickValueProfit

Gets the calculated tick price for a profitable position.

```
double  TickValueProfit() const
```

Return Value

The calculated tick price for a profitable position.

Note

The symbol should be selected by Name method.

TickValueLoss

Gets the calculated tick price for a losing position.

```
double  TickValueLoss() const
```

Return Value

The calculated tick price for a losing position.

Note

The symbol should be selected by Name method.

TickSize

Gets the minimal change of price.

```
double  TickSize() const
```

Return Value

Minimal change of price.

Note

The symbol should be selected by Name method.

ContractSize

Gets the amount of trade contract.

```
double  ContractSize() const
```

Return Value

Amount of trade contract.

Note

The symbol should be selected by Name method.

LotsMin

Gets the minimal volume to close a deal.

```
double  LotsMin() const
```

Return Value

Minimal volume to close a deal.

Note

The symbol should be selected by Name method.

LotsMax

Gets the maximal volume to close a deal.

```
double  LotsMax() const
```

Return Value

Maximal volume to close a deal.

Note

The symbol should be selected by Name method.

LotsStep

Gets the minimal step of volume change to close a deal.

```
double  LotsStep() const
```

Return Value

Minimal step of volume change to close a deal.

Note

The symbol should be selected by Name method.

LotsLimit

Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

```
double  LotsLimit() const
```

Return Value

The maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

Note

The symbol should be selected by Name method.

SwapLong

Gets the value of long position swap.

```
double  SwapLong() const
```

Return Value

Value of long position swap.

Note

The symbol should be selected by Name method.

SwapShort

Gets the value of short position swap.

```
double  SwapShort() const
```

Return Value

Value of short position swap.

Note

The symbol should be selected by Name method.

CurrencyBase

Gets the name of symbol base currency.

```
string  CurrencyBase() const
```

Return Value

Name of symbol base currency.

Note

The symbol should be selected by Name method.

CurrencyProfit

Gets the profit currency name.

```
string  CurrencyProfit() const
```

Return Value

Profit currency name.

Note

The symbol should be selected by Name method.

CurrencyMargin

Gets the margin currency name.

```
string  CurrencyMargin() const
```

Return Value

Margin currency name.

Note

The symbol should be selected by Name method.

Bank

Gets the name of current quote source.

```
string  Bank() const
```

Return Value

Name of current quote source.

Note

The symbol should be selected by Name method.

Description

Gets the string description of symbol.

```
string  Description() const
```

Return Value

String description of symbol.

Note

The symbol should be selected by Name method.

Path

Gets the path in symbols tree.

```
string  Path() const
```

Return Value

Gets the path in symbols tree.

Note

The symbol should be selected by Name method.

SessionDeals

Gets the number of deals in the current session.

```
long  SessionDeals() const
```

Return Value

Number of deals in the current session.

Note

The symbol should be selected by Name method.

SessionBuyOrders

Gets the number of Buy orders at the moment.

```
long  SessionBuyOrders() const
```

Return Value

Number of Buy orders at the moment.

Note

The symbol should be selected by Name method.

SessionSellOrders

Gets then number of Sell orders at the moment.

```
long  SessionSellOrders() const
```

Return Value

Number of Sell orders at the moment.

Note

The symbol should be selected by Name method.

SessionTurnover

Gets summary turnover of the current session.

```
double  SessionTurnover() const
```

Return Value

Summary turnover of the current session.

Note

The symbol should be selected by Name method.

SessionInterest

Gets the summary of open interest of the current session.

```
double  SessionInterest() const
```

Return Value

Summary open interest of the current session.

Note

The symbol should be selected by Name method.

SessionBuyOrdersVolume

Gets the current volume of Buy orders.

```
double  SessionBuyOrdersVolume() const
```

Return Value

Current volume of Buy orders.

Note

The symbol should be selected by Name method.

SessionSellOrdersVolume

Gets the current volume of Sell orders.

```
double  SessionSellOrdersVolume() const
```

Return Value

Current volume of Sell orders.

Note

The symbol should be selected by Name method.

SessionOpen

Gets the open price of the current session.

```
double  SessionOpen() const
```

Return Value

Open price of the current session.

Note

The symbol should be selected by Name method.

SessionClose

Gets the close price of the current session.

```
double  SessionClose() const
```

Return Value

Close price of the current session.

Note

The symbol should be selected by Name method.

SessionAW

Gets the average weighted price of the current session.

```
double  SessionAW() const
```

Return Value

Average weighted price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceSettlement

Gets the settlement price of the current session.

```
double  SessionPriceSettlement() const
```

Return Value

Settlement price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceLimitMin

Gets the minimal price of the current session.

```
double  SessionPriceLimitMin() const
```

Return Value

Minimal price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceLimitMax

Gets the maximal price of the current session.

```
double  SessionPriceLimitMax() const
```

Return Value

Maximal price of the current session.

Note

The symbol should be selected by Name method.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_SYMBOL_INFO_INTEGER  prop_id,      // property ID
   long&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property from ENUM_SYMBOL_INFO_INTEGER enumeration.

var

[out] Reference to long type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by Name method.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_SYMBOL_INFO_DOUBLE  prop_id,      // property ID
   double&                  var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property from ENUM_SYMBOL_INFO_DOUBLE enumeration.

var

[out] Reference to double type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by Name method.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_SYMBOL_INFO_STRING  prop_id,      // property ID
   string&                  var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property.

var

[out] Reference to string type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The symbol should be selected by Name method.

NormalizePrice

Returns the value of price normalized using the symbol properties.

```
double  NormalizePrice(
   double  price    // price
   ) const
```

Parameters

price

[in] Price.

Return Value

Normalized price.

Note

The symbol should be selected by Name method.

COrderInfo

COrderInfo is a class for easy access to the pending order properties.

Description

COrderInfo class provides access to the pending order properties.

Declaration

```
   class COrderInfo : public CObject
```

Title

```
   #include <Trade\OrderInfo.mqh>
```

Inheritance hierarchy

CObject

COrderInfo

Class methods by groups

Access to integer type properties	
Ticket	Gets the ticket of an order, previously selected for access
TimeSetup	Gets the time of order placement
TimeSetupMsc	Receives the time of placing an order in milliseconds since 01.01.1970
OrderType	Gets the order type
OrderTypeDescription	Gets the order type as a string
State	Gets the order state
StateDescription	Gets the order state as a string
TimeExpiration	Gets the time of order expiration
TimeDone	Gets the time of order execution or cancellation
TimeDoneMsc	Receives order execution or cancellation time in milliseconds since 01.01.1970
TypeFilling	Gets the type of order execution by remainder
TypeFillingDescription	Gets the type of order execution by remainder as a string
TypeTime	Gets the type of order at the time of the expiration
TypeTimeDescription	Gets the order type by expiration time as a string
Magic	Gets the ID of expert that placed the order

Access to integer type properties	
PositionId	Gets the ID of position
Access to double type properties	
VolumeInitial	Gets the initial volume of order
VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of a Limit order
Access to text properties	
Symbol	Gets the name of order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
State	
StoreState	Saves the order parameters
CheckState	Checks the current parameters against the saved parameters
Selection	
Select	Selects an order by ticket for further access to its properties
SelectByIndex	Selects an order by index for further access to its properties

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Ticket

Gets the ticket of an order.

```
ulong  Ticket() const
```

Return Value

Order ticket if successful, otherwise - ULONG_MAX.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeSetup

Gets the time of order placement.

```
datetime  TimeSetup() const
```

Return Value

Time of order placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeSetupMsc

Receives the time of placing an order for execution in milliseconds since 01.01.1970.

```
ulong  TimeSetupMsc() const
```

Return Value

The time of placing an order for execution in milliseconds since 01.01.1970.

Note

Order should be preliminarily selected for access using Select (by ticket) or SelectByIndex (by index) method.

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE  OrderType()
```

Return Value

Order type from ENUM_ORDER_TYPE enumeration.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the order type as a string.

```
string  TypeDescription() const
```

Return Value

Order type as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE  State() const
```

Return Value

Order state from ENUM_ORDER_STATE enumeration.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StateDescription

Gets the order state as a string.

```
string  StateDescription() const
```

Return Value

Order state as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeExpiration

Gets the order expiration time.

```
datetime  TimeExpiration() const
```

Return Value

Order expiration time, set on its placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime  TimeDone() const
```

Return Value

Time of order execution or cancellation.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeDoneMsc

Receives order execution or cancellation time in milliseconds since 01.01.1970.

```
ulong  TimeDoneMsc() const
```

Return Value

Order execution or cancellation time in milliseconds since 01.01.1970.

Note

Order should be preliminarily selected for access using Select (by ticket) or SelectByIndex (by index) method.

TypeFilling

Gets the order filling type.

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

Return Value

Order filling type from ENUM_ORDER_TYPE_FILLING enumeration.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeFillingDescription

Gets the order filling type as a string.

```
string  TypeFillingDescription() const
```

Return Value

Order filling type as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

Return Value

Type of order at the time of the expiration from ENUM_ORDER_TYPE_TIME enumeration.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string  TypeTimeDescription() const
```

Return Value

Order type by expiration time as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of an Expert Advisor that placed the order.

```
long  Magic() const
```

Return Value

ID of an Expert Advisor that placed the order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position.

```
long  PositionId() const
```

Return Value

ID of position, in which the order was involved.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double  VolumeInitial() const
```

Return Value

Initial volume of order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double  VolumeCurrent() const
```

Return Value

Unfilled volume of order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the order price.

```
double  PriceOpen() const
```

Return Value

Price of order placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the order's Stop Loss.

```
double  StopLoss() const
```

Return Value

Order's Stop Loss.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the order's Take Profit.

```
double  TakeProfit() const
```

Return Value

Order's Take Profit.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price by order symbol.

```
double  PriceCurrent() const
```

Return Value

Current price by order symbol.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceStopLimit

Gets the price of a pending order.

```
double  PriceStopLimit() const
```

Return Value

Pending order price.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of order symbol.

```
string  Symbol() const
```

Return Value

Name of order symbol.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the order comment.

```
string  Comment() const
```

Return Value

Order comment.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,      // property ID
   long&                        var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property from ENUM_ORDER_PROPERTY_INTEGER enumeration.

var

[out] Reference to long type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property from ENUM_ORDER_PROPERTY_DOUBLE enumeration.

var

[out] Reference to double type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,      // property ID
   string&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of string property from ENUM_ORDER_PROPERTY_STRING enumeration.

var

[out] Reference to string type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StoreState

Saves the order parameters.

```
void  StoreState()
```

Return Value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool  CheckState()
```

Return Value

true - if the order parameters have changed since the last call of the StoreState() method, otherwise - false.

Select

Selects an order by ticket for further access to its properties.

```
bool  Select(
   ulong  ticket    // order ticket
   )
```

Return Value

true – success, false – unable to select order.

SelectByIndex

Selects an order by index.

```
bool  SelectByIndex(
   int   index    // order index
   )
```

Parameters

index

[in] Order index.

Return Value

true - success, false - unable to select order.

CHistoryOrderInfo

CHistoryOrderInfo is a class for easy access to the history order properties.

Description

CHistoryOrderInfo class provides easy access to the history order properties.

Declaration

```
class CHistoryOrderInfo : public CObject
```

Title

```
#include <Trade\HistoryOrderInfo.mqh>
```

Inheritance hierarchy

CObject
    CHistoryOrderInfo

Class methods by groups

Access to integer type properties	
TimeSetup	Gets the time of order placement
TimeSetupMsc	Receives the time of placing an order in milliseconds since 01.01.1970
OrderType	Gets the order type
OrderTypeDescription	Gets the order type as a string
State	Gets the order state
StateDescription	Gets the order state as a string
TimeExpiration	Gets the time of order expiration
TimeDone	Gets the time of order execution or cancellation
TimeDoneMsc	Receives order execution or cancellation time in milliseconds since 01.01.1970
TypeFilling	Gets the type of order execution by remainder
TypeFillingDescription	Gets the type of order execution by remainder as a string
TypeTime	Gets the type of order at the time of the expiration
TypeTimeDescription	Gets the order type by expiration time as a string
Magic	Gets the ID of expert that placed the order
PositionId	Gets the ID of position

Access to integer type properties	
Access to double type properties	
VolumeInitial	Gets the initial volume of order
VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of a Limit order
Access to text properties	
Symbol	Gets the order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets the ticket/selects the order
SelectByIndex	Selects the order by index

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

TimeSetup

Gets the time of order placement.

```
datetime  TimeSetup() const
```

Return Value

Time of order placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeSetupMsc

Receives the time of placing an order for execution in milliseconds since 01.01.1970.

```
ulong  TimeSetupMsc() const
```

Return Value

The time of placing an order for execution in milliseconds since 01.01.1970.

Note

Historical order should be preliminarily selected for access using Ticket (by ticket) or SelectByIndex (by index) method.

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE  OrderType() const
```

Return Value

Order type from ENUM_ORDER_TYPE enumeration.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the order type as a string.

```
string  TypeDescription() const
```

Return Value

Order type as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE  State() const
```

Return Value

Order state from ENUM_ORDER_STATE enumeration.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

StateDescription

Gets the order state as a string.

```
string  StateDescription() const
```

Return Value

Order state as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeExpiration

Gets the time of order expiration.

```
datetime  TimeExpiration() const
```

Return Value

Time of order expiration, set on its placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime  TimeDone() const
```

Return Value

Time of order execution or cancellation.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeDoneMsc

Receives order execution or cancellation time in milliseconds since 01.01.1970.

```
ulong  TimeDoneMsc() const
```

Return Value

Order execution or cancellation time in milliseconds since 01.01.1970.

Note

Historical order should be preliminarily selected for access using Ticket (by ticket) or SelectByIndex (by index) method.

TypeFilling

Gets the type of order execution by remainder.

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

Return Value

Type of order execution by remainder from ENUM_ORDER_TYPE_FILLING enumeration.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeFillingDescription

Gets the type of order execution by remainder as a string.

```
string  TypeFillingDescription() const
```

Return Value

Type order of execution by remainder as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

Return Value

Type of order at the time of the expiration from ENUM_ORDER_TYPE_TIME enumeration.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string  TypeTimeDescription() const
```

Return Value

Order type by expiration time as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of the Expert Advisor, that placed the order.

```
long  Magic() const
```

Return Value

ID of the Expert Advisor, that placed the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position.

```
long  PositionId() const
```

Return Value

ID of position, in which the order was involved.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double  VolumeInitial() const
```

Return Value

Initial volume of order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double  VolumeCurrent() const
```

Return Value

Unfilled volume of order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the order price.

```
double  PriceOpen() const
```

Return Value

Price of order placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the Stop Loss price of the order.

```
double  StopLoss() const
```

Return Value

Stop Loss price of the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the Take Profit price of the order.

```
double  TakeProfit() const
```

Return Value

The Take Profit price of the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price of the order's symbol.

```
double  PriceCurrent() const
```

Return Value

The current price of order's symbol.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceStopLimit

Gets the pending order price.

```
double  PriceStopLimit() const
```

Return Value

Pending orders price.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of order symbol.

```
string  Symbol() const
```

Return Value

Name of order symbol.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the order comment.

```
string  Comment() const
```

Return Value

Order comment.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,      // property ID
   long&                        var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property from ENUM_ORDER_PROPERTY_INTEGER enumeration.

var

[out] Reference to long type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property from ENUM_ORDER_PROPERTY_DOUBLE enumeration.

var

[out] Reference to double type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,      // property ID
   string&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property from ENUM_ORDER_PROPERTY_STRING enumeration.

var

[out] Reference to string type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Ticket (Get method)

Gets the order ticket.

```
ulong  Ticket() const
```

Return Value

Order ticket.

Ticket (Set method)

Select the order for further work.

```
void  Ticket(
   ulong  ticket      // ticket
   )
```

Parameters

ticket

[in] Order ticket.

SelectByIndex

Selects a historical order by index.

```
bool  SelectByIndex(
   int   index    // order index
   )
```

Parameters

index

[in] Historical order index.

Return Value

true - success, false - unable to select order.

CPositionInfo

CPositionInfo is a class for easy access to the open position properties.

Description

CPositionInfo class provides easy access to the open position properties.

Declaration

```
   class CPositionInfo : public CObject
```

Title

```
   #include <Trade\PositionInfo.mqh>
```

Inheritance hierarchy

CObject
    CPositionInfo

Class methods by groups

Access to integer type properties	
Time	Gets the time of position opening
TimeMsc	Receives the time of position opening in milliseconds since 01.01.1970
TimeUpdate	Receives the time of position changing in seconds since 01.01.1970
TimeUpdateMsc	Receives the time of position changing in milliseconds since 01.01.1970
PositionType	Gets the position type
TypeDescription	Gets the position type as a string
Magic	Gets the ID of expert, that opened the position
Identifier	Gets the ID of position
Access to double type properties	
Volume	Gets the volume of position
PriceOpen	Gets the price of position opening
StopLoss	Gets the price of position's Stop Loss
TakeProfit	Gets the price of position's Take Profit

Access to integer type properties	
PriceCurrent	Gets the current price by position symbol
Commission	Gets the amount of commission by position
Swap	Gets the amount of swap by position
Profit	Gets the amount of current profit by position
Access to text properties	
Symbol	Gets the name of position symbol
Comment	Gets the comment of the position
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Selection	
Select	Selects the position
SelectByIndex	Selects the position by index
SelectByMagic	Selects a position with the specified symbol name and magic number
SelectByTicket	Selects the position by ticket
State	
StoreState	Saves the position parameters
CheckState	Checks the current parameters against the saved parameters

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Time

Gets the time of position opening.

```
datetime  Time() const
```

Return Value

Time of position opening.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeMsc

Receives position opening time in milliseconds since 01.01.1970.

```
ulong  TimeMsc() const
```

Return Value

Position opening time in milliseconds since 01.01.1970.

Note

Position should be preliminarily selected for access using Select (by symbol) or SelectByIndex (by index) method.

TimeUpdate

Receives the time of position changing in seconds since 01.01.1970.

```
datetime  TimeUpdate() const
```

Return Value

Time of position changing in seconds since 01.01.1970.

Note

Position should be preliminarily selected for access using Select (by symbol) or SelectByIndex (by index) method.

TimeUpdateMsc

Receives the time of position changing in milliseconds since 01.01.1970.

```
ulong  TimeUpdateMsc() const
```

Return Value

The time of position changing in milliseconds since 01.01.1970.

Note

Position should be preliminarily selected for access using Select (by symbol) or SelectByIndex (by index) method.

PositionType

Gets the position type.

```
ENUM_POSITION_TYPE  PositionType() const
```

Return Value

Position type from ENUM_POSITION_TYPE enumeration.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the position type as a string.

```
string  TypeDescription() const
```

Return Value

Position type as a string.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of Expert Advisor that opened the position.

```
long  Magic() const
```

Return Value

ID of the Expert Advisor that opened the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Identifier

Gets the ID of position.

```
long  Identifier() const
```

Return Value

ID of position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Volume

Gets the volume of position.

```
double  Volume() const
```

Return Value

Volume of position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the price of position opening.

```
double  PriceOpen() const
```

Return Value

Position open price.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the Stop Loss price of the position.

```
double  StopLoss() const
```

Return Value

The Stop Loss price of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the Take Profit price of the position.

```
double  TakeProfit() const
```

Return Value

The Take Profit price of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price by position symbol.

```
double  PriceCurrent() const
```

Return Value

Current price by position symbol.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Commission

Gets the amount of commission of the position.

```
double  Commission() const
```

Return Value

Amount of commission of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Swap

Gets the amount of swap of the position.

```
double  Swap() const
```

Return Value

Amount of swap of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Profit

Gets the amount of current profit of the position.

```
double  Profit() const
```

Return Value

Amount of current profit of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of position symbol.

```
string  Symbol() const
```

Return Value

Name of position symbol.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the comment of the position.

```
string  Comment() const
```

Return Value

Comment of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_POSITION_PROPERTY_INTEGER  prop_id,      // property ID
   long&                           var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property from ENUM_POSITION_PROPERTY_INTEGER enumeration.

var

[out] Reference to long type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                        var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property from ENUM_POSITION_PROPERTY_DOUBLE enumeration.

var

[in] Reference to double type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_POSITION_PROPERTY_STRING  prop_id,      // property ID
   string&                        var           // reference to variable
   )  const
```

Parameters

prop_id

[in] ID of text property from ENUM_POSITION_PROPERTY_STRING enumeration.

var

[out] Reference to string type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Select

Select the position for further work.

```
bool  Select(
   const string  symbol      // symbol
   )
```

Parameters

symbol

[in] Symbol for position selection.

SelectByIndex

Selects the position by index for further access to its properties.

```
bool  SelectByIndex(
   int   index    // position index
   );
```

Return Value

true - success, false - unable to select position.

SelectByMagic

Select a position based on the name of a financial instrument and magic number to further work with.

```
bool  SelectByMagic(
   const string  symbol, // symbol name
   const ulong   magic   // magic number
   );
```

Parameters

symbol

[in] Symbol name.

magic

[in] Magic number of the position.

Return Value

true - successful, false - unable to select position.

SelectByTicket

Selects position by ticket for further operation.

```
bool  SelectByTicket(
   ulong  ticket      // position ticket
   );
```

Parameters

ticket

[in] Position ticket.

Return value

true - success, false - no position selected.

StoreState

Saves the position parameters.

```
void  StoreState()
```

Return Value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool  CheckState()
```

Return Value

true - the position parameters have changed since the last call of the StoreState() method, otherwise - false.

CDealInfo

CDealInfo is a class for easy access to the deal properties.

Description

CDealInfo class provides access to the deal properties.

Declaration

```
class CDealInfo : public CObject
```

Title

```
#include <Trade\DealInfo.mqh>
```

Inheritance hierarchy

CObject
    CDealInfo

Class methods by groups

Access to integer type properties	
Order	Gets the order by which the deal is executed
Time	Gets the time of deal execution
TimeMsc	Receives the time of a deal execution in milliseconds since 01.01.1970
DealType	Gets the deal type
TypeDescription	Gets the deal type as a string
Entry	Gets the deal direction
EntryDescription	Gets the deal direction as a string
Magic	Gets the ID of expert, that executed the deal
PositionId	Gets the ID of position, in which the deal was involved
Access to double type properties	
Volume	Gets the volume of deal
Price	Gets the deal price
Commision	Gets the amount of commission of the deal
Swap	Gets the amount of swap when position is closed
Profit	Gets the financial result of deal

Access to integer type properties	
Access to text properties	
Symbol	Gets the name of deal symbol
Comment	Gets the deal comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets ticket/selects the deal
SelectByIndex	Selects the deal by index

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Order

Gets the order by which the deal is executed.

```
long  Order() const
```

Return Value

Order by which the deal is executed.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Time

Gets the time of deal execution.

```
datetime  Time() const
```

Return Value

Time of deal execution.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeMsc

Receives the time of a deal execution in milliseconds since 01.01.1970.

```
ulong  TimeMsc() const
```

Return Value

The time of a deal execution in milliseconds since 01.01.1970.

Note

Deal should be preliminarily selected for access using Ticket (by ticket) or SelectByIndex (by index) method.

DealType

Gets the deal type.

```
ENUM_DEAL_TYPE  DealType() const
```

Return Value

Deal type from ENUM_DEAL_TYPE enumeration.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the deal type as a string.

```
string  TypeDescription() const
```

Return Value

Deal type as a string.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Entry

Compares the deal direction.

```
ENUM_DEAL_ENTRY  Entry() const
```

Return Value

Deal direction (value of ENUM_DEAL_ENTRY enumeration).

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

EntryDescription

Gets the deal direction as a string.

```
string  EntryDescription() const
```

Return Value

Deal direction as a string.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of the Expert Advisor, that executed the deal.

```
long  Magic() const
```

Return Value

ID of the Expert Advisor, that executed the deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position, in which the deal was involved.

```
long  PositionId() const
```

Return Value

ID of position, in which the deal was involved.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Volume

Gets the volume of deal.

```
double  Volume() const
```

Return Value

Volume of deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Price

Gets the deal price.

```
double  Price() const
```

Return Value

Deal price.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Commission

Gets the amount of commission of the deal.

```
double  Commission() const
```

Return Value

Amount of commission of the deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Swap

Gets the amount of swap when position is closed.

```
double  Swap() const
```

Return Value

Amount of swap when position is closed.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Profit

Gets the financial result of the deal.

```
double  Profit() const
```

Return Value

Financial result of the deal (in deposit currency).

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of the deal symbol.

```
string  Symbol() const
```

Return Value

Name of the deal symbol.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the deal comment.

```
string  Comment() const
```

Return Value

Deal comment.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_DEAL_PROPERTY_INTEGER  prop_id,      // property ID
   long&                       var           // reference to variable
   )  const
```

Parameters

prop_id

[in] ID of integer type property from ENUM_DEAL_PROPERTY_INTEGER enumeration.

var

[out] Reference to long type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_DEAL_PROPERTY_DOUBLE  prop_id,     // property ID
   double&                    var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property from ENUM_DEAL_PROPERTY_DOUBLE enumeration.

var

[out] Reference to double type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_DEAL_PROPERTY_STRING  prop_id,      // property ID
   string&                    var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property from ENUM_DEAL_PROPERTY_STRING enumeration.

var

[out] Reference to string type variable to place result.

Return Value

true - success, false - unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Ticket (Get method)

Gets the deal ticket.

```
ulong  Ticket() const
```

Return Value

Deal ticket.

Ticket (Set method)

Select the position for further work.

```
void  Ticket(
   ulong  ticket      // ticket
   )
```

Parameters

ticket

[in] Deal ticket.

SelectByIndex

Selects the deal by index for further access to its properties.

```
bool  SelectByIndex(
   int   index    // order index
   )
```

Return Value

true - success, false - unable to select the deal.

CTrade

CTrade is a class for easy access to the trade functions.

Description

CTrade class provides easy access to the trade functions.

Declaration

```
class CTrade : public CObject
```

Title

```
#include <Trade\Trade.mqh>
```

Inheritance hierarchy

CObject

  CTrade

Direct descendants

  CExpertTrade

Class methods by groups

Setting parameters	
LogLevel	Sets logging level
SetExpertMagicNumber	Sets the expert ID
SetDeviationInPoints	Sets the allowed deviation
SetTypeFilling	Sets filling type of the order
SetTypeFillingBySymbol	Sets filling type of the order according to the specified symbol settings
SetAsyncMode	Sets asynchronous mode for trade operations
SetMarginMode	Sets margin calculation mode in accordance with the current account settings
Operations with orders	
OrderOpen	Places a pending order with specified parameters
OrderModify	Modifies the pending order parameters
OrderDelete	Deletes a pending order
Operations with positions	
PositionOpen	Opens a position with specified parameters

Setting parameters	
PositionModify	Modifies position parameters by the specified symbol or position ticket
PositionClose	Closes a position for the specified symbol
PositionClosePartial	Partially closes a position on a specified symbol or having a specified ticket
PositionCloseBy	Closes a position with the specified ticket by an opposite position
Additional methods	
Buy	Opens a long position with specified parameters
Sell	Opens a short position with specified parameters
BuyLimit	Places a pending order of the Buy Limit type with specified parameters
BuyStop	Places a pending order of the Buy Stop type with specified parameters
SellLimit	Places a pending order of the Sell Limit type with specified parameters
SellStop	Places a pending order of the Sell Stop type with specified parameters
Access to the last request parameters	
Request	Gets the copy of the last request structure
RequestAction	Gets the trade operation type
RequestActionDescription	Gets the trade operation type as string
RequestMagic	Gets the magic number of the Expert Advisor
RequestOrder	Gets the order ticket used in the last request
RequestSymbol	Gets the name of the symbol used in the last request
RequestVolume	Gets the trade volume (in lots) used in the last request
RequestPrice	Gets the price used in the last request
RequestStopLimit	Gets the price of pending order of Stop Limit type used in the last request
RequestSL	Gets the Stop Loss price of the order used in the last request
RequestTP	Gets the Take Profit price of the order used in the last request
RequestDeviation	Gets the maximum allowable price deviation of the order used in the last request
RequestType	Gets the type of the order used in the last request

Setting parameters	
RequestTypeDescription	Gets the type of the order (as string) used in the last request
RequestTypeFilling	Gets the filling type of the order used in the last request
RequestTypeFillingDescription	Gets the filling type of the order (as string) used in the last request
RequestTypeTime	Gets the validity period of the order used in the last request
RequestTypeTimeDescription	Gets the validity period of the order (as string) used in the last request
RequestExpiration	Gets the expiration time of the order used in the last request
RequestComment	Gets the comment of the order used in the last request
RequestPosition	Gets position ticket
RequestPositionBy	Gets opposite position ticket
Access to the last request checking results	
CheckResult	Gets the copy of the structure of the last request check result.
CheckResultRetcode	Gets the value of the retcode field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultRetcodeDescription	Gets the string description of the retcode field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultBalance	Gets the value of the balance field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultEquity	Gets the value of the equity field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultProfit	Gets the value of the floating profit after executing a trading operation.
CheckResultMargin	Gets the value of the margin field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultMarginFree	Gets the value of the margin_free field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultMarginLevel	Gets the value of the margin_level field of MqlTradeCheckResult type, filled while checking the request correctness
CheckResultComment	Gets the value of the comment field of MqlTradeCheckResult type, filled while checking the request correctness
Access to the last request execution results	
Result	Gets the copy of the structure of the last request result

Setting parameters	
ResultRetcode	Gets the code of request result
ResultRetcodeDescription	Gets the code of request result as a string
ResultDeal	Gets the deal ticket
ResultOrder	Gets the order ticket
ResultVolume	Gets the volume of deal or order
ResultPrice	Gets the price, confirmed by broker
ResultBid	Gets the current bid price (the requote)
ResultAsk	Gets the current ask price (the requote)
ResultComment	Gets the broker comment
Auxiliary methods	
PrintRequest	Prints the last request parameters into journal
PrintResult	Prints the results of the last request into journal
FormatRequest	Prepares the formatted string with last request parameters
FormatRequestResult	Prepares the formatted string with results of the last request execution

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

LogLevel

Sets logging level for messages.

```
void  LogLevel(
   ENUM_LOG_LEVELS  log_level      // level
   )
```

Parameters

log_level

[in] New logging level.

Return Value

None.

Note

LOG_LEVEL_NO and less disables displaying any messages (set up automatically in the optimization mode). LOG_LEVEL_ERRORS enables displaying only error messages (value by default). LOG_LEVEL_ALL and greater enables displaying any messages (set up automatically in the test mode).

ENUM_LOG_LEVELS

Identifier	Description	Value
LOG_LEVEL_NO	Displaying messages disabled	0
LOG_LEVEL_ERRORS	Only error messages are displayed	1
LOG_LEVEL_ALL	All messages are displayed	2

SetExpertMagicNumber

Sets the expert ID.

```
void  SetExpertMagicNumber(
   ulong  magic      // ID
   )
```

Parameters

magic

[in] New ID of the expert.

Return Value

None.

SetDeviationInPoints

Sets the allowed deviation.

```
void  SetDeviationInPoints(
   ulong  deviation      // deviation
   )
```

Parameters

deviation

[in] Allowed deviation in points.

Return Value

None.

SetTypeFilling

Sets filling type of the order.

```
void  SetTypeFilling(
   ENUM_ORDER_TYPE_FILLING  filling      // order filling type
   )
```

Parameters

filling

[in] Order filling type from ENUM_ORDER_TYPE_FILLING enumeration.

Return Value

None.

SetTypeFillingBySymbol

Sets filling type of the order according to the specified symbol settings.

```
bool  SetTypeFillingBySymbol(
   const string   symbol      // symbol name
   )
```

Parameters

symbol

[in] Name of the symbol, in which SYMBOL_FILLING_MODE contains allowed order filling policies.

Return Value

true – successful execution, false – failed to define the filling policy.

Note

If SYMBOL_FILLING_FOK and SYMBOL_FILLING_IOC filling policies are allowed for a symbol simultaneously, the ORDER_FILLING_FOK value is set for the order.

SetAsyncMode

Sets asynchronous mode for trade operations.

```
void  SetAsyncMode(
   bool  mode      // asynchronous mode flag
   )
```

Parameters

mode

[in] Asynchronous mode flag.

Return Value

None.

Note

This mode is used for asynchronous (without waiting for the trade server's response to a sent request) trade operations (see OrderSendAsync).

SetMarginMode

Sets margin calculation mode in accordance with the current account settings.

```
void  SetMarginMode()
```

Return Value

No.

Note

The margin calculation mode is specified in ENUM_ACCOUNT_MARGIN_MODE.

OrderOpen

Places the pending order with set parameters.

```
bool  OrderOpen(
   const string          symbol,           // symbol
   ENUM_ORDER_TYPE       order_type,       // order type
   double                volume,           // order volume
   double                limit_price,      // StopLimit price
   double                price,            // execution price
   double                sl,               // Stop Loss price
   double                tp,               // Take Profit price
   ENUM_ORDER_TYPE_TIME  type_time,        // type by expiration
   datetime              expiration,       // expiration
   const string          comment=""        // comment
   )
```

Parameters

symbol

[in] Name of trade instrument.

order_type

[in] Type of order trade operation from ENUM_ORDER_TYPE enumeration.

volume

[in] Requested order volume.

limit_price

[in] Price at which the StopLimit order will be placed.

price

[in] Price at which the order must be executed.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

type_time

[in] Order type by execution from ENUM_ORDER_TYPE_TIME enumeration.

expiration

[in] Expiration date of pending order.

comment=""

[in] Order comment.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderSend(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of a trade request (trade server return code) using ResultRetcode() and value returned by ResultOrder().

OrderModify

Modifies the pending order parameters.

```
bool  OrderModify(
   ulong                 ticket,         // order ticket
   double                price,          // execution price
   double                sl,             // Stop Loss price
   double                tp,             // Take Profit price
   ENUM_ORDER_TYPE_TIME  type_time,      // type by expiration
   datetime              expiration,     // expiration
   double                stoplimit       // Limit order price
   )
```

Parameters

ticket

[in] Order ticket.

price

[in] The new price by which the order must be executed (or the previous value, if the change is not necessary).

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

type_time

[in] The new type of order by expiration from ENUM_ORDER_TYPE_TIME enumeration (or the previous value, if the change is not necessary).

expiration

[in] The new expiration date of pending order (or the previous value, if the change is not necessary).

stoplimit

[in] New price used for setting a Limit order when the price reaches *price* value. It is specified only for StopLimit orders.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderModify(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode().

OrderDelete

Deletes the pending order.

```
bool  OrderDelete(
   ulong  ticket      // order ticket
   )
```

Parameters

ticket

[in] Order ticket.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderDelete(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode().

PositionOpen

Opens a position with the specified parameters.

```
bool  PositionOpen(
   const string      symbol,          // symbol
   ENUM_ORDER_TYPE   order_type,      // order type to open position
   double            volume,          // position volume
   double            price,           // execution price
   double            sl,              // Stop Loss price
   double            tp,              // Take Profit price
   const string      comment=""       // comment
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to open position.

order_type

[in] Order type (trade operation) to open position from ENUM_ORDER_TYPE enumeration.

volume

[in] Requested position volume.

price

[in] Price at which the position must be opened.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

comment=""

[in] Position comment.

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionOpen(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultDeal().

PositionModify

Modifies the position parameters by specified symbol.

```
bool  PositionModify(
   const string  symbol,      // symbol
   double        sl,          // Stop Loss price
   double        tp           // Take Profit price
   )
```

Modifies position parameters by the specified ticket.

```
bool  PositionModify(
   const ulong   ticket,      // position ticket
   double        sl,          // Stop Loss price
   double        tp           // Take Profit price
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to modify position.

ticket

[in] Ticket of the position to be modified.

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionModify(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode().

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol. In this case, PositionModify will modify the position with the lowest ticket.

PositionClose

Closes a position by the specified symbol.

```
bool  PositionClose(
   const string  symbol,                  // symbol
   ulong         deviation=ULONG_MAX      // deviation
   )
```

Closes a position with the specified ticket.

```
bool  PositionClose(
   const ulong   ticket,                  // position ticket
   ulong         deviation=ULONG_MAX      // deviation
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to close position.

ticket

[in] Ticket of a closed position.

deviation=ULONG_MAX

[in] Maximal deviation from the current price (in points).

Return Value

true - successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionClose(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode().

For the "netting" interpretation of positions (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), only one position can exist for a symbol at any moment of time. This position is a result of one or more deals. Do not confuse positions with valid pending orders, which are also displayed on the Trading tab of the Toolbox window.

If individual positions are allowed (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be open for one symbol. In this case, PositionClose will close a position with the lowest ticket.

PositionClosePartial

Partially closes a position on a specified symbol in case of a "hedging" accounting.

```
bool  PositionClosePartial(
   const string  symbol,                 // symbol
   const double  volume,                 // volume
   ulong         deviation=ULONG_MAX      // deviation
   )
```

Partially closes a position having a specified ticket in case of a "hedging" accounting.

```
bool  PositionClosePartial(
   const ulong   ticket,                 // position ticket
   const double  volume,                 // volume
   ulong         deviation=ULONG_MAX      // deviation
   )
```

Parameters

symbol

[in] Name of a trading instrument, on which a position is closed partially. If a symbol (not a ticket) is specified for a position partial closing, the first detected position having a specified MagicNumber (Expert Advisor ID) on the symbol is selected. Therefore, it is sometimes better to use PositionClosePartial() with a specified position ticket.

volume

[in] Volume, by which a position should be decreased. If the value exceeds the volume of a partially closed position, it is closed in full. No position in the opposite direction is opened.

ticket

[in] Closed position ticket.

deviation=ULONG_MAX

[in] The maximum deviation from the current price (in points).

Return Value

true if the basic check of structures is successful, otherwise false.

Note

A successful completion of the PositionClosePartial(...) method does not always mean a successful execution of a trading operation. You should call the ResultRetcode() method to check the result of a trade request (trade server return code).

In the "netting" system (ACCOUNT_MARGIN_MODE_RETAIL_NETTING and ACCOUNT_MARGIN_MODE_EXCHANGE), for each symbol, at any given moment of time, only one position can be open, which is the result of one or more deals. Do not confuse the current pending orders with positions that are also displayed on the "Trade" tab of the client terminal's "Toolbox" panel.

In case of the position representation (ACCOUNT_MARGIN_MODE_RETAIL_HEDGING), multiple positions can be opened on each symbol simultaneously. In this case, PositionClose closes a position having a least ticket.

PositionCloseBy

Closes a position with the specified ticket by an opposite position.

```
bool  PositionCloseBy(
   const ulong   ticket,        // position ticket
   const ulong   ticket_by      // opposite position ticket
   )
```

Parameters

ticket

[in] Ticket of the closed position.

ticket_by

[in] Ticket of the opposite position used for closing.

Returned value

true - the basic check of structures is successful, otherwise - false.

Note

A successful completion of the PositionCloseBy(...) method does not always mean a successful execution of a trading operation. You should call the ResultRetcode() method to check the result of a trade request (trade server return code).

Buy

Opens a long position with specified parameters.

```
bool  Buy(
   double         volume,          // position volume
   const string   symbol=NULL,     // symbol
   double         price=0.0,       // execution price
   double         sl=0.0,          // stop loss price
   double         tp=0.0,          // take profit price
   const string   comment=""       // comment
   )
```

Parameters

volume

[in] Requested position volume.

symbol=NULL

[in] Position symbol. If it is not specified, the current symbol will be used.

price=0.0

[in] Price. If the price is not specified, the current market Ask price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the Buy(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultDeal().

Sell

Opens a short position with specified parameters.

```
bool  Sell(
   double          volume,           // position volume
   const string    symbol=NULL,      // symbol
   double          price=0.0,        // execution price
   double          sl=0.0,           // stop loss price
   double          tp=0.0,           // take profit price
   const string    comment=""        // comment
   )
```

Parameters

volume

[in] Requested position volume.

symbol=NULL

[in] Position symbol. If the symbol is not specified, the current symbol will be used.

price=0.0

[in] Price. If the price is not specified, the current market Bid price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the Sell(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultDeal().

BuyLimit

Places the pending order of Buy Limit type (buy at the price lower than current market price) with specified parameters.

```
bool  BuyLimit(
   double              volume,                        // order volume
   double              price,                         // order price
   const string        symbol=NULL,                   // symbol
   double              sl=0.0,                        // stop loss price
   double              tp=0.0,                        // take profit price
   ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC,      // order lifetime
   datetime            expiration=0,                  // order expiration time
   const string        comment=""                     // comment
   )
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from ENUM_ORDER_TYPE_TIME enumeration.

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the BuyLimit(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultOrder().

BuyStop

Places the pending order of Buy Stop type (buy at the price higher than current market price) with specified parameters.

```
bool  BuyStop(
   double                volume,                      // order volume
   double                price,                       // order price
   const string          symbol=NULL,                 // symbol
   double                sl=0.0,                      // stop loss price
   double                tp=0.0,                      // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,    // order lifetime
   datetime              expiration=0,                // order expiration time
   const string          comment=""                   // comment
   )
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from ENUM_ORDER_TYPE_TIME enumeration.

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the BuyStop(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultOrder().

SellLimit

Places the pending order of Sell Limit type (sell at the price higher than current market price) with specified parameters.

```
bool  SellLimit(
   double                volume,                      // order volume
   double                price,                       // order price
   const string          symbol=NULL,                 // symbol
   double                sl=0.0,                      // stop loss price
   double                tp=0.0,                      // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,    // order lifetime
   datetime              expiration=0,                // order expiration time
   const string          comment=""                   // comment
   )
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from ENUM_ORDER_TYPE_TIME enumeration.

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the SellLimit(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultOrder().

SellStop

Places the pending order of Sell Stop type (sell at the price lower than current market price) with specified parameters.

```
bool  SellStop(
   double                volume,                        // order volume
   double                price,                         // order price
   const string          symbol=NULL,                   // symbol
   double                sl=0.0,                        // stop loss price
   double                tp=0.0,                        // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,      // order lifetime
   datetime              expiration=0,                  // order expiration time
   const string          comment=""                     // comment
   )
```

Parameters

volume

[in] Requested order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol is not specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime from ENUM_ORDER_TYPE_TIME enumeration.

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Return Value

true - successful check of the structures, otherwise - false.

Note

Successful completion of the SellStop(...) method does not always mean successful execution of the trade operation. It is necessary to check the result of trade request (trade server return code) using ResultRetcode() and value returned by ResultOrder().

Request

Gets the copy of the last request structure.

```
void  Request(
   MqlTradeRequest&  request      // reference
   ) const
```

Parameters

request

[out] Reference to the copied structure.

Return Value

None.

RequestAction

Gets the trade operation type.

```
ENUM_TRADE_REQUEST_ACTIONS  RequestAction() const
```

Return Value

Trade operation type used in the last request from ENUM_TRADE_REQUEST_ACTIONS enumeration.

RequestActionDescription

Gets the trade operation type as string.

```
string  RequestActionDescription() const
```

Return Value

Trade operation type (as string) used in the last request.

RequestMagic

Gets the magic number of the Expert Advisor.

```
ulong  RequestMagic() const
```

Return Value

The magic number (ID) of the Expert Advisor, used in the last request.

RequestOrder

Gets the order ticket.

```
ulong  RequestOrder() const
```

Return Value

Order ticket of the last request.

RequestSymbol

Gets the name of the symbol.

```
string  RequestSymbol() const
```

Return Value

The name of the symbol used in the last request.

RequestVolume

Gets the trade volume (in lots).

```
double  RequestVolume() const
```

Return Value

The requested trade volume (in lots) used in the last request.

RequestPrice

Gets the order execution price.

```
double  RequestPrice() const
```

Return Value

Order execution price used in the last request.

RequestStopLimit

Gets the price of pending order of Stop Limit type.

```
double  RequestStoplimit() const
```

Return Value

The price of pending order of Stop Limit type used in the last request.

RequestSL

Gets the Stop Loss price of an order.

```
double  RequestSL() const
```

Return Value

The Stop Loss price used in the last request.

RequestTP

Gets the Take Profit price of the order.

```
double  RequestTP() const
```

Return Value

The Take Profit price used in the last request.

RequestDeviation

Gets the maximum price deviation from a requested price.

```
ulong  RequestDeviation() const
```

Return Value

The maximum allowable price deviation from a requested price used in the last request.

RequestType

Gets the type of the order.

```
ENUM_ORDER_TYPE  RequestType() const
```

Return Value

Order type used in the last request from ENUM_ORDER_TYPE enumeration.

RequestTypeDescription

Gets the type of the order (as string).

```
string  RequestTypeDescription() const
```

Return Value

The order type (as string) used in the last request.

RequestTypeFilling

Gets the filling type of the order.

```
ENUM_ORDER_TYPE_FILLING  RequestTypeFilling() const
```

Return Value

The filling type of the order from ENUM_ORDER_TYPE_FILLING used in the last request.

RequestTypeFillingDescription

Gets the filling type of the order (as string).

```
string  RequestTypeFillingDescription() const
```

Return Value

The filling type (as string) of the order used in the last request.

RequestTypeTime

Gets the validity period of the order.

```
ENUM_ORDER_TYPE_TIME  RequestTypeTime() const
```

Return Value

The validity period of the order from ENUM_ORDER_TYPE_TIME enumeration used in the last request.

RequestTypeTimeDescription

Gets the validity period of the order (as string).

```
string  RequestTypeTimeDescription() const
```

Return Value

The validity period of the order (as string) used in the last request.

RequestExpiration

Gets the expiration time of a pending order.

```
datetime  RequestExpiration() const
```

Return Value

The expiration time of a pending order used in the last request.

RequestComment

Gets the comment of the order.

```
string  RequestComment() const
```

Return Value

The comment of the order used in the last request.

RequestPosition

Gets position ticket.

```
ulong  RequestPosition() const
```

Return value

Position ticket used in the last request.

RequestPositionBy

Gets opposite position ticket.

```
ulong  RequestPositionBy() const
```

Return value

Opposite position ticket used in the last request.

Result

Gets the copy of the structure of the last request result.

```
void  Result(
   MqlTradeResult&   result      // reference
   ) const
```

Parameters

result

[out] Reference to the structure of MqlTradeResult type for copying.

Return Value

None.

ResultRetcode

Gets the code of request execution result.

```
uint  ResultRetcode() const
```

Return Value

The code of request result.

ResultRetcodeDescription

Gets the code of request execution result as a string.

```
string  ResultRetcodeDescription() const
```

Return Value

Code of the last request result as a string.

ResultDeal

Gets the deal ticket.

```
ulong  ResultDeal() const
```

Return Value

Deal ticket if the deal is executed.

ResultOrder

Gets the order ticket.

```
ulong  ResultOrder() const
```

Return Value

Order ticket if the order is placed.

ResultVolume

Gets the volume of deal or order.

```
double  ResultVolume() const
```

Return Value

Volume of deal or order confirmed by a broker.

ResultPrice

Gets the price confirmed by a broker.

```
double  ResultPrice() const
```

Return Value

Price confirmed by a broker.

ResultBid

Gets the current bid price (the requote).

```
double  ResultBid() const
```

Return Value

Current bid price (the requote).

ResultAsk

Gets the current ask price (the requote).

```
double  ResultAsk() const
```

Return Value

Current ask price (the requote).

ResultComment

Gets the broker comment.

```
string  ResultComment() const
```

Return Value

Broker comment to the operation.

CheckResult

Gets the copy of the structure of the last request check result.

```
void  CheckResult(
   MqlTradeCheckResult&  check_result      // reference
   ) const
```

Parameters

check_result

[out] Reference to the target structure of the MqlTradeCheckResult type for copying.

Return Value

None.

CheckResultRetcode

Gets the value of the retcode field of MqlTradeCheckResult type, filled while checking the request correctness.

```
uint  CheckResultRetcode() const
```

Return Value

The value of the retcode field (error code) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultRetcodeDescription

Gets the string description of the retcode field of MqlTradeCheckResult type filled while checking the request correctness.

```
string  ResultRetcodeDescription() const
```

Return Value

The string description of the retcode field (Error code) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultBalance

Gets the value of the balance field of MqlTradeCheckResult type filled while checking the request correctness.

```
double  CheckResultBalance() const
```

Return Value

The value of the balance field (balance value that will be after the execution of the trade operation) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultEquity

Gets the value of the equity field of MqlTradeCheckResult type filled while checking the request correctness.

```
double  CheckResultEquity() const
```

Return Value

The value of the equity field (equity value that will be after the execution of the trade operation) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultProfit

Gets the value of a floating profit.

```
double  CheckResultProfit() const
```

Return Value

The value of a floating profit after executing a trading operation.

CheckResultMargin

Gets the value of the margin field of MqlTradeCheckResult type filled while checking the request correctness.

```
double  CheckResultMargin() const
```

Return Value

The value of the margin field (margin required for the trade operation) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultMarginFree

Gets the value of the margin_free field of MqlTradeCheckResult type filled while checking the request correctness.

```
double  CheckResultMarginFree() const
```

Return Value

The value of the margin_free field (free margin that will be left after the execution of the trade operation) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultMarginLevel

Gets the value of the margin_level field of MqlTradeCheckResult type filled while checking the request correctness.

```
double  CheckResultMarginLevel() const
```

Return Value

The value of the margin_level field (margin level that will be set after the execution of the trade operation) of MqlTradeCheckResult type filled while checking the request correctness.

CheckResultComment

The value of the comment field of MqlTradeCheckResult type filled while checking the request correctness.

```
string  CheckResultComment() const
```

Return Value

The value of the comment field (comment to the reply code, error description) of MqlTradeCheckResult type filled while checking the request correctness.

PrintRequest

Prints the last request parameters into journal.

```
void  PrintRequest() const
```

Return Value

None.

PrintResult

Prints the results of the last request into journal.

```
void  PrintResult() const
```

Return Value

None.

FormatRequest

Prepares the formatted string with last request parameters.

```
string  FormatRequest(
   string&                  str,         // string
   const MqlTradeRequest&   request      // request
   ) const
```

Parameters

str

[in] Target string passed by reference.

request

[in] A structure of MqlTradeRequest type with parameters of the last request.

Return Value

None.

FormatRequestResult

Prepares the formatted string with results of the last request execution.

```
string  FormatRequestResult(
   string&                    str,          // string
   const MqlTradeRequest&   request,      // request
   const MqlTradeResult&    result        // result
   ) const
```

Parameters

str

[in] Target string passed by reference.

request

[in] A structure of MqlTradeRequest type with parameters of the last request.

result

[in] A structure of MqlTradeResult type with results of the last request.

Return Value

None.

CTerminalInfo

CTerminalInfo is a class for simplified access to the properties of mql5 program environment.

Description

CTerminalInfo class provides access to the properties of mql5 program environment.

Declaration

```
   class CTerminalInfo : public CObject
```

Title

```
   #include <Trade\TerminalInfo.mqh>
```

Inheritance hierarchy

CObject

    CTerminalInfo

Class methods by groups

Methods for access to the properties of integer type	
Build	Gets the build number of the client terminal
IsConnected	Gets the information about connection to trade server
IsDLLsAllowed	Gets the information about permission of DLL usage
IsTradeAllowed	Gets the information about permission to trade
IsEmailEnabled	Gets the information about permission to send e-mails to SMTP server and login, specified in the terminal settings
IsFtpEnabled	Gets the information about permission to send trade reports to FTP server and login, specified in the terminal settings
MaxBars	Gets the information about maximum number of bars on chart
CodePage	Gets the information about the code page of the language in the client terminal
CPUCores	Gets the information about the CPU cores
MemoryPhysical	Gets the information about the physical memory (in Mb)
MemoryTotal	Gets the information about the total memory available for the terminal/agent process (in Mb)
MemoryAvailable	Gets the information about the free memory available for the terminal/agent process (in Mb)

Methods for access to the properties of integer type	
MemoryUsed	Gets the information about the memory used by the terminal/agent process (in Mb)
IsX64	Gets the information about the type of the client terminal
OpenCLSupport	Gets the information about the version of OpenCL supported by video card
DiskSpace	Gets the information about free disk space (in Mb)
Methods for access to the properties of string type	
Language	Gets the language of the client terminal
Name	Gets the name of the client terminal
Company	Gets the company name of the client terminal
Path	Gets the folder of the client terminal
DataPath	Gets the data folder of the client terminal
CommonDataPath	Gets the common data folder of all client terminals, installed on the computer
Access to MQL5 API functions	
InfoInteger	Gets the value of the property of integer type
InfoString	Gets the value of property of string type

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Build

Gets the build number of the client terminal.

```
int  CBuild() const
```

Return Value

Build number of the client terminal.

Note

The terminal build number is defined by TerminalInfoInteger() function (TERMINAL_BUILD property).

IsConnected

Gets the information about connection to trade server.

```
bool  IsConnected() const
```

Return Value

true - the terminal is connected to a trade server, otherwise - false.

Note

Connection status is defined by TerminalInfoInteger() function (TERMINAL_CONNECTED property).

IsDLLsAllowed

Gets the information about permission of DLL usage.

```
bool  IsDLLsAllowed() const
```

Return Value

true - DLL usage is allowed, otherwise - false.

Note

Permission of DLL usage is defined by TerminalInfoInteger() function (TERMINAL_DLLS_ALLOWED property).

IsTradeAllowed

Gets the information about permission to trade.

```
bool  IsTradeAllowed() const
```

Return Value

true - trade allowed, otherwise - false.

Note

Permission to trade is defined by TerminalInfoInteger() function (TERMINAL_TRADE_ALLOWED property).

IsEmailEnabled

Gets the information about permission to send e-mails to SMTP server and login specified in the terminal settings.

```
bool  IsEmailEnabled() const
```

Return Value

true - sending e-mails is allowed, otherwise - false.

Note

Permission to send e-mails is defined by TerminalInfoInteger() function (TERMINAL_EMAIL_ENABLED property).

IsFtpEnabled

Gets the information about permission to send trade reports to FTP server and login specified in the terminal settings.

```
bool  IsFtpEnabled() const
```

Return Value

true - sending trade reports to FTP server is allowed, otherwise - false.

Note

Permission to send trade reports is defined TerminalInfoInteger() function (TERMINAL_FTP_ENABLED property).

MaxBars

Gets the maximum number of bars on chart specified in the terminal settings.

```
int  MaxBars() const
```

Return Value

Maximum number of bars on the chart.

Note

The maximum number of bars on chart is defined by TerminalInfoInteger() function (TERMINAL_MAXBARS property).

CodePage

Gets the information about code page of the language in the terminal.

```
int  CodePage() const
```

Return Value

Code page of the language in the client terminal.

Note

Code page of the language is defined by TerminalInfoInteger() function (TERMINAL_CODEPAGE property).

CPUCores

Gets the information about the number of CPU cores in the system.

```
int  CPUCores() const
```

Return Value

Number of CPU cores in the system.

Note

The number of CPU cores is defined by TerminalInfoInteger() function (TERMINAL_CPU_CORES property).

MemoryPhysical

Gets the information about the physical memory (in Mb).

```
int  MemoryPhysical() const
```

Return Value

Physical memory (in Mb).

Note

The physical memory is defined by TerminalInfoInteger() function (TERMINAL_MEMORY_PHYSICAL property).

MemoryTotal

Gets the information about the total memory available to the terminal/agent (in Mb).

```
int  MemoryTotal() const
```

Return Value

Total memory (in Mb) available to the terminal/agent.

Note

The total memory available to the terminal/agent is defined by TerminalInfoInteger() function (TERMINAL_MEMORY_TOTAL property).

MemoryAvailable

Gets the information about the free memory available to the client terminal/agent (in Mb).

```
int  MemoryTotal() const
```

Return Value

Free memory (in Mb) available to the terminal/agent.

Note

The free memory available to the client terminal/agent is defined by TerminalInfoInteger() function (TERMINAL_MEMORY_TOTAL property).

MemoryUsed

Gets the information about the memory used by the client terminal/agent (in Mb).

```
int  MemoryUsed() const
```

Return Value

The memory used by the client terminal/agent (in Mb).

Note

The memory used by the client terminal/agent is defined by TerminalInfoInteger() function (TERMINAL_MEMORY_USED property).

IsX64

Gets the information about the type of the client terminal.

```
bool  IsX64() const
```

Return Value

true - 64-bit version is used, otherwise - false.

Note

The type of the terminal is defined by TerminalInfoInteger() function (TERMINAL_X64 property).

OpenCLSupport

Gets the information about the version of OpenCL supported by video card.

```
int  OpenCLSupport() const
```

Return Value

OpenCL version having the following form: 0x00010002 = "1.2". 0 means that OpenCL is not supported.

Note

OpenCL version is defined by TerminalInfoInteger() function (TERMINAL_OPENCL_SUPPORT property).

DiskSpace

Gets the information about free disk space available for MQL5\Files folder of the terminal/agent (in Mb).

```
int  MDiskSpace() const
```

Return Value

Free disk space.

Note

Free disk space is defined by TerminalInfoInteger() function (TERMINAL_DISK_SPACE property).

Language

Gets the information about the language of the client terminal.

```
string  Language() const
```

Return Value

Language used in the client terminal.

Note

The terminal language is defined by TerminalInfoString() function (TERMINAL_LANGUAGE property).

Name

Gets the information on the name of the client terminal.

```
string  Name() const
```

Return Value

Name of the client terminal.

Note

The name of the terminal is defined by TerminalInfoString() function (TERMINAL_NAME property).

Company

Gets the information about the name of the comopany.

```
string  Company() const
```

Return Value

The name of the company.

Note

The company name is defined by TerminalInfoString() function (TERMINAL_COMPANY property).

Path

Gets the client terminal folder.

```
string  Path() const
```

Return Value

The client terminal folder.

Note

The client terminal folder is used by TerminalInfoString() function (TERMINAL_PATH property).

DataPath

Gets the information about the terminal data folder.

```
string  DataPath() const
```

Return Value

Data folder of the client terminal.

Note

Client terminal data folder is defined by TerminalInfoString() function (TERMINAL_DATA_PATH property).

CommonDataPath

Gets the common data folder of all client terminals installed on the computer.

```
string  CommonDataPath() const
```

Return Value

Common data folder of all installed terminals.

Note

The common data folder of all installed terminals is defined by TerminalInfoString() function (COMMON_DATA_PATH property).

InfoInteger

Returns the value of a corresponding integer property of the mql5 program environment.

```
int  TerminalInfoInteger(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. It can be one of the values of the enumeration ENUM_TERMINAL_INFO_INTEGER.

Return Value

Value of int type.

Note

The property value is defined by TerminalInfoInteger() function.

InfoString

The function returns the value of a corresponding property of the mql5 program environment. The property must be of string type.

```
string  TerminalInfoString(
   int  property_id      // property ID
   );
```

Parameters

property_id

[in] Identifier of a property. It can be one the values of the enumeration ENUM_TERMINAL_INFO_STRING.

Return Value

Value of string type.

Note

The property value is defined by TerminalInfoString() function.

Classes for Creating and Testing Trading Strategies

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and especially testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert folder.

Base classes	Description
CExpertBase	Base class for all trading strategy classes
CExpert	Base class for Expert Advisor
CExpertSignal	Base class for Trading Signal classes
CExpertTrailing	Base class for Trailing Stop classes
CExpertMoney	Base class for Money Management classes

Trading signal classes	Description
CSignalAC	The module of signals based on market models of the indicator Accelerator Oscillator.
CSignalAMA	The module of signals based on market models of the indicator Adaptive Moving Average.
CSignalAO	The module of signals based on market models of the indicator Awesome Oscillator.
CSignalBearsPower	The module of signals based on market models of the oscillator Bears Power.
CSignalBullsPower	The module of signals based on market models of the oscillator Bulls Power.
CSignalCCI	The module of signals based on market models of the oscillator Commodity Channel Index.
CSignalDeM	The module of signals based on market models of the oscillator DeMarker.
CSignalDEMA	The module of signals based on market models of the indicator Double Exponential Moving Average.
CSignalEnvelopes	The module of signals based on market models of the indicator Envelopes.
CSignalFrAMA	The module of signals based on market models of the indicator Fractal Adaptive Moving Average.
CSignalITF	The module of filtration of signals by time.

Trading signal classes	Description
CSignalMACD	The module of signals based on market models of the oscillator MACD.
CSignalMA	The module of signals based on market models of the indicator Moving Average.
CSignalSAR	The module of signals based on market models of the indicator Parabolic SAR.
CSignalRSI	The module of signals based on market models of the oscillator Relative Strength Index.
CSignalRVI	The module of signals based on market models of the oscillator Relative Vigor Index.
CSignalStoch	The module of signals based on market models of the oscillator Stochastic.
CSignalTRIX	The module of signals based on market models of the oscillator Triple Exponential Average.
CSignalTEMA	The module of signals based on market models of the indicator Triple Exponential Moving Average.
CSignalWPR	The module of signals based on market models of the oscillator Williams Percent Range.

Trailing Stop classes	Description
CTrailingFixedPips	This class implements Trailing Stop algorithm based on fixed points
CTrailingMA	This class implements Trailing Stop algorithm based on the values of Moving Average indicator
CTrailingNone	A stub class, it does not use any Trailing Stop algorithm
CTrailingPSAR	This class implements Trailing Stop algorithm based on the values of Parabolic SAR indicator

Money Management classes	Description
CMoneyFixedLot	A class with an algorithm based on trading with predefined fixed lot size.
CMoneyFixedMargin	A class with an algorithm based on trading with predefined fixed margin.
CMoneyFixedRisk	A class with an algorithm based on trading with predefined risk.
CMoneyNone	A class with an algorithm based on trading with minimal allowed lot size.
CMoneySizeOptimized	A class with an algorithm based on trading with variable lot size, depending on the results of the previous deals.

Base classes for Expert Advisors

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and especially testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies classes) is placed in the terminal directory, in the Include\Expert folder.

Class	Description
CExpertBase	Base class for all trading strategy classes
CExpert	Base class for Expert Advisor
CExpertSignal	Base class for Trading Signal classes
CExpertTrailing	Base class for Trailing Stop classes
CExpertMoney	Base class for Money Management classes

CExpertBase

CExpertBase is a base class for the CExpert class and all auxiliary trading strategy classes.

Description

CExpertBase provides the data and methods, which are common to all objects of the Expert Advisor.

Declaration

```
class CExpertBase : public CObject
```

Title

```
#include <Expert\ExpertBase.mqh>
```

Inheritance hierarchy

CObject
    CExpertBase

Direct descendants

CExpert, CExpertMoney, CExpertSignal, CExpertTrailing

Class Methods by Groups

Public Methods:

Initialization	
virtual Init	Initializes the object
virtual ValidationSettings	Checks the settings
Parameters	
Symbol	Sets the symbol
Period	Sets the timeframe
Magic	Sets the Expert Advisor ID
Indicators and Timeseries	
virtual SetPriceSeries	Sets pointers to external timeseries (price series)
virtual SetOtherSeries	Sets pointers to external timeseries (non-price series)
virtual InitIndicators	Initializes the indicators and timeseries
Access to Protected Data	
InitPhase	Gets the current phase of object initialization
TrendType	Sets trend type

Initialization	
UsedSeries	Gets bitmask of timeseries used
EveryTick	Sets the "Every tick" flag
Access to Timeseries	
Open	Gets the element of the Open timeseries by index
High	Gets the element of the High timeseries by index
Low	Gets the element of the Low timeseries by index
Close	Gets the element of the Close timeseries by index
Spread	Gets the element of the Spread timeseries by index
Time	Gets the element of the Time timeseries by index
TickVolume	Gets the element of the TickVolume timeseries by index
RealVolume	Gets the element of the RealVolume timeseries by index

Protected Methods:

Initialization of Timeseries	
InitOpen	Open timeseries initialization method
InitHigh	High timeseries initialization method
InitLow	Low timeseries initialization method
InitClose	Close timeseries initialization method
InitSpread	Spread timeseries initialization method
InitTime	Time timeseries initialization method
InitTickVolume	TickVolume timeseries initialization method
InitRealVolume	RealVolume timeseries initialization method
Service Methods	
virtual PriceLevelUnit	Gets the price level unit
virtual StartIndex	Gets the index of starting bar to analyze
virtual CompareMagic	Compares the Expert Advisor ID with the specified value

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

InitPhase

Gets the current phase of the object initialization.

```
ENUM_INIT_PHASE  InitPhase()
```

Return Value

Current phase of the object initialization.

Note

The object initialization consist of several phases:

1. Start initialization.

- start - after finish of the constructor
- finish - after successful completion of the Init(...) method.
- allowed - call of the Init(...) method
- not allowed - call of the ValidationSettings() method and other initialization methods

2. Parameters setting phase. In this phase you need to set all the object parameters, used for creation of indicators.

- start - after successful completion of the Init(...) method
- finish - after successful completion of the ValidationSettings() method
- allowed - call of Symbol(...) and Period(...) methods
- not allowed - call of the Init(...), SetPriceSeries(...), SetOtherSeries(...) and InitIndicators(...) methods

3. Checking of parameters.

- start - after successful completion of the ValidationSettings() method
- finish - after successful completion of the InitIndicators(...) method
- allowed - call of the Symbol(...), Period(...) and InitIndicators(...) methods
- not allowed - call of any other initialization methods

4. Finish of initialization.

- start - after successful completion of the InitIndicators(...) method
- not allowed - call of initialization methods

TrendType

Sets trend type.

```
void  TrendType(
   M_TYPE_TREND     value          // value
   )
```

Parameters

value

[in] New value of trend type.

Return Value

None.

UsedSeries

Gets the bitmask of timeseries used.

```
int  UsedSeries()
```

Return Value

The list of used timeseries as bitmask.

Note

If the bit is set, the corresponding timeseries is used, if it is not set, the timeseries is not used.

The bit-timeseries correspondence:

bit 0 - Open timeseries,
bit 1 - High timeseries,
bit 2 - Low timeseries,
bit 3 - Close timeseries,
bit 4 - Spread timeseries,
bit 5 - Time timeseries,
bit 6 - TickVolume timeseries,
bit 7 - RealVolume timeseries.

EveryTick

Sets the "Every tick" flag.

```
void  EveryTick(
   bool     value          // flag
   )
```

Parameters

value

[in] New value of a flag.

Return Value

None.

Note

If the flag is set, each price (tick) change at a working symbol is processed.

If the flag is not set, the processing method is called only at a new bar on the working timeframe and symbol.

Open

Gets the element of the Open timeseries by index.

```
double  Open(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Open timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

High

Gets the element of the High timeseries by index.

```
double  High(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the High timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Low

Gets the element of the Low timeseries by index.

```
double  Low(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Low timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Close

Gets the element of the Close timeseries by index.

```
double  Close(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Close timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Spread

Gets the element of the Spread timeseries by index.

```
double  Spread(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Spread timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Time

Gets the element of the Time timeseries by index.

```
datetime  Time(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the Time timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

TickVolume

Gets the element of the TickVolume timeseries by index.

```
long  TickVolume(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the TickVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

RealVolume

Gets the element of the RealVolume timeseries by index.

```
long  RealVolume(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Return Value

If successful, it returns the numerical value of the RealVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Init

Initializes the object.

```
bool  Init(
   CSymbolInfo       symbol,    // symbol
   ENUM_TIMEFRAMES  period,    // timeframe
   double            point      // point
   )
```

Parameters

symbol

[in] Pointer to the object of CSymbolInfo type for access to symbol information.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

point

[in] The "weight" of 2/4-digit point.

Return Value

true - successful completion, otherwise - false.

Symbol

Sets the object symbol.

```
bool  Symbol(
   string     name      // symbol
   )
```

Parameters

name

[in] Symbol.

Return Value

true - successful, otherwise - false.

Note

The setting of working symbol is necessary if the object uses the symbol different from symbol defined at the first initialization.

Period

Sets the object timeframe.

```
bool  Period(
   ENUM_TIMEFRAMES  value     // timeframe
   )
```

Parameters

value

[in] Timeframe.

Return Value

true - successful, otherwise - false.

Note

The setting of working timeframe is necessary if the object uses the timeframe different from timeframe defined at the initialization.

Magic

Sets the Expert Advisor ID.

```
void  Magic(
   ulong  value      // magic
   )
```

Parameters

value

[in] Expert Advisor ID.

Return Value

None.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

SetPriceSeries

Sets pointers to external price series.

```
virtual bool  SetPriceSeries(
   CiOpen*     open,          // pointer
   CiHigh*     high,          // pointer
   CiLow*      low,           // pointer
   CiClose*    close          // pointer
   )
```

Parameters

open

[in] Pointer to Open timeseries.

high

[in] Pointer to High timeseries.

low

[in] Pointer to Low timeseries.

close

[in] Pointer to Close timeseries.

Return Value

true - successful, otherwise - false.

Note

The setting of pointers to external timeseries (price series) is necessary if the object uses the symbol and timeframe (set during the first initialization) and price timeseries necessary for further work.

SetOtherSeries

Sets pointers to external non-price series.

```
virtual bool  SetOtherSeries(
   CiSpread*        spread,         // pointer
   CiTime*          time,           // pointer
   CiTickVolume*    tick_volume,    // pointer
   CiRealVolume*    real_volume     // pointer
   )
```

Parameters

spread

[in] Pointer to Spread timeseries.

time

[in] Pointer to Time timeseries.

tick_volume

[in] Pointer to TickVolume timeseries.

real_volume

[in] Pointer to RealVolume timeseries.

Return Value

true - successful, otherwise - false.

Note

The setting of pointers to external timeseries (price series) is necessary if the object uses the symbol and timeframe (set during the first initialization) and price timeseries necessary for further work.

InitIndicators

Initializes all indicators and time series.

```
virtual bool  InitIndicators(
   CIndicators*   indicators=NULL     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The timeseries are initialized only if the object uses the symbol or timeframe different from the symbol or timeframe defined at initialization.

InitOpen

Initalizes the Open timeseries.

```
bool  InitOpen(
   CIndicators*  indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Open timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitHigh

Initalizes the High timeseries.

```
bool  InitHigh(
   CIndicators*  indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The High timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitLow

Initalizes the Low timeseries.

```
bool  InitLow(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Low timeseries is initialized only if the object uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitClose

Initalizes the Close timeseries.

```
bool  InitClose(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Close timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitSpread

Initalizes the Spread timeseries.

```
bool  InitSpread(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Spread timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitTime

Initalizes the Time timeseries.

```
bool  InitTime(
   CIndicators*  indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The Time timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitTickVolume

Initalizes the TickVolume timeseries.

```
bool  InitTickVolume(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The TickVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

InitRealVolume

Initalizes the RealVolume timeseries.

```
bool  InitRealVolume(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful, otherwise - false.

Note

The RealVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe different from the symbol/timeframe defined at the first initialization (and timeseries is used further).

PriceLevelUnit

Gets the price level unit.

```
virtual double  PriceLevelUnit()
```

Return Value

The value of price level unit.

Note

The method of a base class returns the "weight" of the 2/4 digits point.

StartIndex

Gets the index of starting bar to analyze.

```
virtual int  StartIndex()
```

Return Value

The index of starting bar to analyze.

Note

The method returns 0 if the flag to analyze current bar is set to true (analysis from the current bar). If the flag is not set, it returns 1 (analysis from the last completed bar).

CompareMagic

Compares the Expert Advisor ID (magic) with the specified value.

```
virtual bool  CompareMagic(
   ulong  magic     // identifier
   )
```

Parameters

magic

[in] Identifier value to compare.

Return Value

true - identifier matches the specified one, otherwise - false.

CExpert

CExpert is a base class for trading strategies.

It already has some elementary trading "skills". It has built-in algorithms for working with time series and indicators and a set of virtual methods for trading strategy.

How to use it:

1. Prepare an algorithm of the strategy;
2. Create your own class, inherited from CExpert class;
3. Override the virtual methods in your class with your own algorithms.

Description

The CExpert class is a set of virtual methods for implementation of trading strategies.

Note

A position is recognized as belonging to an Expert Advisor and managed by it based on the pair of properties m_symbol and m_magic. In the "hedging" mode, multiple positions can be opened for the same symbol, therefore the m_magic value is important.

Declaration

```
   class CExpert : public CExpertBase
```

Title

```
   #include <Expert\Expert.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpert

Class Methods by Groups

Initialization	
Init	Class instance initialization method
virtual InitSignal	Initializes Trading Signal object
virtual InitTrailing	Initializes Trailing Stop object
virtual InitMoney	Initializes Money Management object
virtual InitTrade	Initializes Trade object
virtual ValidationSettings	Checks the settings
virtual InitIndicators	Initializes indicators and timeseries

Initialization	
virtual InitParameters	Expert Advisor parameters initialization method
virtual Deinit	Class instance deinitialization method
virtual DeinitSignal	Deinitializes Trading Signal object
virtual DeinitTrailing	Deinitializes Trailing Stop object
virtual DeinitMoney	Deinitializes Money Management object
virtual DeinitTrade	Deinitializes Trade object
virtual DeinitIndicators	Deinitializes indicators and timeseries
Parameters	
Magic	Sets the Expert Advisor ID
MaxOrders	Gets/sets the maximum amount of allowed orders
OnTickProcess	Sets a flag to proceed the "OnTick" event
OnTradeProcess	Sets a flag to proceed the "OnTrade" event
OnTimerProcess	Sets a flag to proceed the "OnTimer" event
OnChartEventProcess	Sets a flag to proceed the "OnChartEvent" event
OnBookEventProcess	Sets a flag to proceed the "OnBookEvent" event
Event Processing Methods	
OnTick	OnTick event handler
OnTrade	OnTrade event handler
OnTimer	OnTimer event handler
OnChartEvent	OnChartEvent event handler
OnBookEvent	OnBookEvent event handler
Update Methods	
Refresh	Updates all data
Processing	
Processing	Main processing algorithm
Market Entry Methods	
CheckOpen	Checks position opening conditions
CheckOpenLong	Checks conditions to open long position
CheckOpenShort	Checks conditions to open short position
OpenLong	Opens a long position

Initialization	
OpenShort	Opens a short position
Market Exit Methods	
CheckClose	Checks conditions to close current position
CheckCloseLong	Checks conditions to close long position
CheckCloseShort	Checks conditions to close short position
CloseAll	Closes the opened position and deletes all orders
Close	Closes the opened position
CloseLong	Closes the long position
CloseShort	Closes the short position
Position Reverse Methods	
CheckReverse	Checks conditions to reverse opened position
CheckReverseLong	Checks conditions to reverse long position
CheckReverseShort	Checks conditions to reverse short position
ReverseLong	Performs reverse operation of long position
ReverseShort	Performs reverse operation of short position
Position/Order Trailing Methods	
CheckTrailingStop	Checks conditions to modify position parameters
CheckTrailingStopLong	Checks Trailing Stop conditions of long position
CheckTrailingStopShort	Checks Trailing Stop conditions of short position
TrailingStopLong	Performs Trailing Stop for long position
TrailingStopShort	Performs Trailing Stop for short position
CheckTrailingOrderLong	Checks Trailing Stop conditions of Buy Limit/Stop order
CheckTrailingOrderShort	Checks Trailing Stop conditions of Sell Limit/Stop order
TrailingOrderLong	Performs Trailing Stop for Buy Limit/Stop order
TrailingOrderShort	Performs Trailing Stop for Sell Limit/Stop order
Order Delete Methods	
CheckDeleteOrderLong	Checks conditions to delete Buy order
CheckDeleteOrderShort	Checks conditions to delete Sell order
DeleteOrders	Deletes all orders
DeleteOrder	Deletes Stop/Limit order

Initialization	
DeleteOrderLong	Deletes Buy Limit/Stop order
DeleteOrderShort	Deletes Sell Limit/Stop order
Trade Volume Methods	
LotOpenLong	Gets trade volume for buy operation
LotOpenShort	Gets trade volume for sell operation
LotReverse	Gets trade volume for position reverse operation
Trade History Methods	
PrepareHistoryDate	Sets starting date for trade history tracking
HistoryPoint	Creates a checkpoint of trade history (saves number of positions, orders, deals and historical orders)
CheckTradeState	Compares the current state with the saved one and calls the corresponding event handler
Event flags	
WaitEvent	Sets the trading event waiting flag
NoWaitEvent	Resets the trading event waiting flag
Trade Event Processing Methods	
TradeEventPositionStopTake	Event handler of the "Position Stop Loss/Take Profit triggered" event
TradeEventOrderTriggered	Event handler of the "Pending Order Triggered" event
TradeEventPositionOpened	Event handler of the "Position Opened" event
TradeEventPositionVolumeChanged	Event handler of the "Position Volume Changed" event
TradeEventPositionModified	Event handler of the "Position Modified" event
TradeEventPositionClosed	Event handler of the "Position Closed" event
TradeEventOrderPlaced	Event handler of the "Pending Order Placed" event
TradeEventOrderModified	Event handler of the "Pending Order Modified" event
TradeEventOrderDeleted	Event handler of the "Pending Order Deleted" event
TradeEventNotIdentified	Event handler of the non-identified event
Service methods	
TimeframeAdd	Adds a timeframe to track
TimeframesFlags	Forms timeframe flags
SelectPosition	Selects a position to work with

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

Init

Class instance initialization method.

```
bool  Init(
   string              symbol,        // symbol
   ENUM_TIMEFRAMES     period,        // timeframe
   bool                every_tick,    // flag
   ulong               magic          // Expert Advisor identifier
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe from ENUM_TIMEFRAMES enumeration.

every_tick

[in] Flag.

magic

[in] Expert Advisor ID (Magic number).

Return Value

None.

Note

If every_tick is set to true, the Processing() method is called at each tick of the working symbol. otherwise, the Processing() is called only when a new bar is formed on the working timeframe of the EA's working symbol.

Magic

Sets the Expert Advisor ID (magic).

```
void  Magic(
   ulong  value      // new value
   )
```

Parameters

value

[in] New value of Expert Advisor ID.

Return Value

None.

Note

If EA's identifier is changed, the same identifier is assigned to all auxiliary objects.

Implementation

```
//+------------------------------------------------------------------+
//| Sets magic number for object and its dependent objects           |
//| INPUT:  value - new value of magic number.                       |
//| OUTPUT: no.                                                      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
void CExpert::Magic(ulong value)
  {
   if(m_trade!=NULL)    m_trade.SetExpertMagicNumber(value);
   if(m_signal!=NULL)   m_signal.Magic(value);
   if(m_money!=NULL)    m_money.Magic(value);
   if(m_trailing!=NULL) m_trailing.Magic(value);
//---
   CExpertBase::Magic(value);
  }
```

InitSignal

Initializes trading signal object.

```
virtual bool  InitSignal(
   CExpertSignal*      signal=NULL,         // pointer
   )
```

Parameters

signal

[in] Pointer to CExpertSignal class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If signal is NULL, the CExpertSignal class will be used for initialization (it does nothing).

InitTrailing

Initializes trailing stop object.

```
virtual bool  InitTrailing(
   CExpertTrailing*      trailing=NULL,         // pointer
   )
```

Parameters

trailing

[in] Pointer to CExpertTrailing class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If trailing is NULL, the ExpertTrailing class will be used for initialization (it does nothing).

InitMoney

Initializes the money management object.

```
virtual bool  InitMoney(
   CExpertMoney*      money=NULL,         // pointer
   )
```

Parameters

money

[in] Pointer to CExpertMoney class object (or its descendant).

Return Value

true - successful, otherwise - false.

Note

If money is NULL, the CExpertMoney class will be used for initialization (it uses the minimum lot).

InitTrade

Initializes the trade object.

```
virtual bool  InitTrade(
   ulong            magic,       // identifier
   CExpertTrade*    trade=NULL   // pointer
   )
```

Parameters

magic

[in] Expert Advisor ID (will be used in trade requests).

trade

[in] Pointer to trade object.

Return Value

true - successful, otherwise - false.

Deinit

Class instance deinitialization method.

```
virtual void Deinit()
```

Return Value

None.

OnTickProcess

Sets the OnTick event handling flag.

```
void  OnTickOProcess(
   bool      value         // flag
   )
```

Parameters

value

[in] OnTick event handling flag.

Return Value

None.

Note

If the flag is true, the OnTick event is handled. By default, the flag is set to true.

OnTradeProcess

Sets the OnTrade event handling flag.

```
void  OnTradeProcess(
   bool      value         // flag
   )
```

Parameters

value

[in] OnTrade event handling flag.

Return Value

None.

Note

If the flag is true, the OnTrade event is handled. By default, the flag is set to false.

OnTimerProcess

Sets the OnTimer event handling flag.

```
void  OnTimerProcess(
   bool      value         // flag
   )
```

Parameters

value

[in] OnTimer event handling flag.

Return Value

None.

Note

If the flag is true, the OnTimer event is handled. By default, the flag is set to false.

OnChartEventProcess

Sets a flag to handle the OnChartEvent event.

```
void  OnChartEventProcess(
   bool      value         // flag
   )
```

Parameters

value

[in] Flag to handle the OnChartEvent event.

Return Value

None.

Note

If the flag is true, the OnChartEvent event is handled. By default, the flag is set to false.

OnBookEventProcess

Sets a flag to handle the OnBookEvent event.

```
void  OnChartEventProcess(
   bool      value         // flag
   )
```

Parameters

value

[in] Flag to handle the OnBookEvent event.

Return Value

None.

Note

If the flag is true, the OnBookEvent event is handled. By default, the flag is set to false.

MaxOrders (Get Method)

Gets the maximum amount of allowed orders.

```
int  MaxOrders()
```

Return Value

Maximum amount of allowed orders.

MaxOrders (Set Method)

Sets the maximum amount of allowed orders.

```
void  MaxOrders(
   int      max_orders         // amount of orders
   )
```

Parameters

max_orders

[in] New value of maximum amount of allowed orders.

Return Value

None.

Note

By default, the maximum amount of allowed orders is equal to 1.

Signal

Gets the pointer to the trade signal object.

```
CExpertSignal*  Signal() const
```

Return Value

Pointer to the trade signal object.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

It also checks the settings of all the Expert Advisor objects.

InitIndicators

Initializes necessary indicators and timeseries.

```
virtual bool  InitIndicators(
   CIndicators*   indicators=NULL     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful completion, otherwise - false.

Note

The timeseries are initialized if the object uses a symbol or timeframe other than the one defined in the initialization.

Indicators and timeseries of all auxiliary EA objects are initialized.

OnTick

OnTick event handler.

```
virtual void  OnTick()
```

Return Value

None.

OnTrade

OnTrade event handler.

```
virtual void  OnTrade()
```

Return Value

None.

OnTimer

OnTimer event handler.

```
virtual void  OnTimer()
```

Return Value

None.

OnChartEvent

OnChartEvent event handler.

```
virtual void  OnChartEvent(
   const int      id,         // event id
   const long&    lparam,     // long type event parameter
   const double   dparam,     // double type event parameter
   const string   sparam      // string type event parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type.

dparam

[in] Event parameter of double type.

sparam

[in] Event parameter of string type.

Return Value

None.

OnBookEvent

OnBookEvent event handler.

```
virtual void  OnBookEvent(
   const string&      symbol        // symbol
)
```

Parameters

symbol

[in] Symbol of OnBookEvent event.

Return Value

None.

InitParameters

Initializes parameters of Expert Advisor.

```
virtual bool  InitParameters()
```

Return Value

true - successful, otherwise - false.

Note

The InitParameters() function of CExpert base class does nothing and always returns true.

DeinitTrade

Deinitializes trade object.

```
virtual void  DeinitTrade()
```

Return Value

None.

DeinitSignal

Deinitializes trade signal object.

```
virtual void  DeinitSignal()
```

Return Value

None.

DeinitTrailing

Deinitializes trailing object.

```
virtual void  DeinitTrailing()
```

Return Value

None.

DeinitMoney

Deinitializes money management object.

```
virtual void  DeinitMoney()
```

Return Value

None.

DeinitIndicators

Deinitializes indicators and timeseries.

```
virtual void  DeinitIndicators()
```

Return Value

None.

Note

Indicators and timeseries of all auxiliary EA objects are deinitialized.

Refresh

Updates all necessary data.

```
virtual bool  Refresh()
```

Return Value

true - further tick processing is needed, otherwise - false.

Note

It allows to determine the need of tick processing. If it is needed, it updates all quotes and timeseries and indicators data and returns true.

Implementation

```
//+------------------------------------------------------------------+
//| Refreshing data for processing                                   |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if successful, false otherwise.                     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Refresh()
  {
   MqlDateTime time;
//--- refresh rates
   if(!m_symbol.RefreshRates()) return(false);
//--- check need processing
   TimeToStruct(m_symbol.Time(),time);
   if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
      if((m_period_flags & TimeframesFlags(time))==0) return(false);
   m_last_tick_time=time;
//--- refresh indicators
   m_indicators.Refresh();
//--- ok
   return(true);
  }
```

Processing

Main processing algorithm.

```
virtual bool  Processing()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It does the following steps:

1. Checks the presence of the opened position on the symbol. If there is no opened position, skip steps 2, 3, and 4.
2. Checks conditions to reverse opened position (CheckReverse() method). If position has been "reversed", exit.
3. Checks conditions to close position (CheckClose() method). If position has been closed, skip step 4.
4. Checks conditions to modify position parameters (CheckTrailingStop() method). If position parameters have been modified, exit.
5. Check the presence of pending orders on the symbol. If there is no any pending order, go to step 9.
6. Checks condition to delete order (CheckDeleteOrderLong() for buy pending orders or CheckDeleteOrderShort() for sell pending orders). If the order has been deleted, go to step 9.
7. Check conditions to modify pending order parameters (CheckTrailingOrderLong() for buy orders or CheckTrailingOrderShort() for sell orders). If the order parameters have been modified, exit.
8. Exit.
9. Checks conditions to open position (CheckOpen() method).

If you want to implement your own algorithm, you need to override the Processing() method of the descendant class.

Implementation

```
//+------------------------------------------------------------------+
//| Main function                                                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if any trade operation processed, false otherwise.  |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Processing()
  {
//--- check if open positions
   if(m_position.Select(m_symbol.Name()))
     {
      //--- open position is available
      //--- check the possibility of reverse the position
      if(CheckReverse()) return(true);
      //--- check the possibility of closing the position/delete pending orders
      if(!CheckClose())
        {
         //--- check the possibility of modifying the position
         if(CheckTrailingStop()) return(true);
         //--- return without operations
         return(false);
        }
     }
//--- check if placed pending orders
   int total=OrdersTotal();
   if(total!=0)
     {
      for(int i=total-1;i>=0;i--)
        {
         m_order.SelectByIndex(i);
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         if(m_order.OrderType()==ORDER_TYPE_BUY_LIMIT || m_order.OrderType()==ORDER_TY
           {
            //--- check the ability to delete a pending order to buy
            if(CheckDeleteOrderLong()) return(true);
            //--- check the possibility of modifying a pending order to buy
            if(CheckTrailingOrderLong()) return(true);
           }
         else
           {
            //--- check the ability to delete a pending order to sell
            if(CheckDeleteOrderShort()) return(true);
            //--- check the possibility of modifying a pending order to sell
            if(CheckTrailingOrderShort()) return(true);
           }
         //--- return without operations
         return(false);
        }
     }
//--- check the possibility of opening a position/setting pending order
   if(CheckOpen()) return(true);
//--- return without operations
   return(false);
  }
```

SelectPosition

Selects a position to work with.

```
void  SelectPosition()
```

Return Value

No.

Implementation

```
//+------------------------------------------------------------------+
//| Position select                                                  |
//| INPUT:  no.                                                      |
//| OUTPUT: no.                                                      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::SelectPosition(void)
  {
   bool res=false;
//---
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
      res=m_position.SelectByMagic(m_symbol.Name(),m_magic);
   else
      res=m_position.Select(m_symbol.Name());
//---
   return(res);
  }
```

CheckOpen

Checks conditions to open a position.

```
virtual bool  CheckOpen()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open long (CheckOpenLong()) and short (CheckOpenShort()) positions.

Implementation

```
//+------------------------------------------------------------------+
//| Check for position open or limit/stop order set                  |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpen()
  {
   if(CheckOpenLong())  return(true);
   if(CheckOpenShort()) return(true);
//--- return without operations
   return(false);
  }
```

CheckOpenLong

Checks necessity and conditions to open long position.

```
virtual bool  CheckOpenLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open a long position (CheckOpenLong() method of Signal object) and does that with the parameters set by Signal object (OpenLong() method) if the conditions are met.

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position open or limit/stop order set             |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for long enter operations
   if(m_signal.CheckOpenLong(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenLong(price,sl,tp));
     }
//--- return without operations
   return(false);
  }
```

CheckOpenShort

Checks necessity and conditions to open a short position.

```
virtual bool  CheckOpenShort()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to open a short position (CheckOpenShort() method of Signal object) and does that with the parameters set by Signal object (OpenShort() method) if the conditions are met.

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position open or limit/stop order set            |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for short enter operations
   if(m_signal.CheckOpenShort(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenShort(price,sl,tp));
     }
//--- return without operations
   return(false);
  }
```

OpenLong

Opens a long position.

```
virtual bool  OpenLong(
   double    price,     // price
   double    sl,        // Stop Loss
   double    tp         // Take Profit
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets trading volume (LotOpenLong(...) method) and opens a long position (Buy() method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Long position open or limit/stop order set                       |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::OpenLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for open
   double lot=LotOpenLong(price,sl);
//--- check lot for open
   if(lot==0.0) return(false);
//---
   return(m_trade.Buy(lot,price,sl,tp));
  }
```

OpenShort

Opens a short position.

```
virtual bool  OpenShort(
   double    price,     // price
   double    sl,        // Stop Loss
   double    tp         // Take Profit
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets trading volume (LotOpenShort() method) and opens a short position (by calling Sell method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Short position open or limit/stop order set                      |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::OpenShort(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for open
   double lot=LotOpenShort(price,sl);
//--- check lot for open
   if(lot==0.0) return(false);
//---
   return(m_trade.Sell(lot,price,sl,tp));
  }
```

CheckReverse

Checks necessity and conditions to reverse an open position.

```
virtual bool  CheckReverse()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse long (CheckReverseLong()) and short (CheckReverseShort()) positions.

Implementation

```
//+------------------------------------------------------------------+
//| Check for position reverse                                       |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverse()
  {
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of reverse the long position
      if(CheckReverseLong())  return(true);
     }
   else
      //--- check the possibility of reverse the short position
      if(CheckReverseShort()) return(true);
//--- return without operations
   return(false);
  }
```

CheckReverseLong

Checks necessity and conditions to reverse a long position.

```
virtual bool  CheckReverseLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse a long position (CheckReverseLong() method of Signal object) and perform reverse operation of the current long position with the parameters set by Signal object (ReverseLong(...) method) if the conditions are met.

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position reverse                                  |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for long reverse operations
   if(m_signal.CheckReverseLong(price,sl,tp,expiration)) return(ReverseLong(price,sl,t
//--- return without operations
   return(false);
  }
```

CheckReverseShort

Checks necessity and conditions to reverse a short position.

```
virtual bool  CheckReverseLong()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks the necessity to reverse a short position (CheckReverseShort() method of Signal object) and perform reverse operation of the current short position with the parameters set by Signal object (ReverseShort() method) if the conditions are met.

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position reverse                                 |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for short reverse operations
   if(m_signal.CheckReverseShort(price,sl,tp,expiration)) return(ReverseShort(price,sl
//--- return without operations
   return(false);
  }
```

ReverseLong

Performs reverse operation of a long position.

```
virtual bool  ReverseLong(
   double     price,     // price
   double     sl,        // Stop Loss
   double     tp         // Take Profit
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets the position reverse volume (LotReverse() method) and reverses a long position (Sell() method of Trade object) if trading volume is not equal to 0.

In the "hedging" mode of position accounting, position reversal is performed as the closure of the existing position and opening of a new opposite one with the remaining volume.

Implementation

```
//+------------------------------------------------------------------+
//| Long position reverse                                            |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::ReverseLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE)
      return(false);
//--- get lot for reverse
   double lot=LotReverse(sl);
//--- check lot
   if(lot==0.0)
      return(false);
//---
   bool result=true;
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
     {
      //--- first close existing position
      lot-=m_position.Volume();
      result=m_trade.PositionCloseByTicket(m_position.Identifier());
     }
   if(result)
      result=m_trade.Sell(lot,price,sl,tp);
//---
   return(result);
  }
```

ReverseShort

Performs reverse operation of a short position.

```
virtual bool  ReverseShort(
   double     price,     // price
   double     sl,        // Stop Loss
   double     tp         // Take Profit
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It gets position reverse volume (LotReverse(...) method) and perform trade operation of the short position reverse (Buy() method of Trade object) if trading volume is not equal to 0.

In the "hedging" mode of position accounting, position reversal is performed as the closure of the existing position and opening of a new opposite one with the remaining volume.

Implementation

```
//+------------------------------------------------------------------+
//| Short position reverse                                           |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::ReverseLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE)
      return(false);
//--- get lot for reverse
   double lot=LotReverse(sl);
//--- check lot
   if(lot==0.0)
      return(false);
//---
   bool result=true;
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
     {
      //--- first close existing position
      lot-=m_position.Volume();
      result=m_trade.PositionCloseByTicket(m_position.Identifier());
     }
   if(result)
      result=m_trade.Sell(lot,price,sl,tp);
//---
   return(result);
  }
```

CheckClose

Checks conditions to close position.

```
virtual bool  CheckClose()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

1. It checks Expert Advisor Stop Out conditions (CheckClose() method of money management object). If condition is satisfied, it closes the position, deletes all orders (CloseAll()), and exits.
2. It checks conditions to close long or short position (CheckCloseLong() or CheckCloseShort() methods) and if position is closed, it deletes all orders (DeleteOrders() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for position close or limit/stop order delete              |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckClose()
  {
   double lot;
//--- position must be selected before call
   if((lot=m_money.CheckClose(GetPointer(m_position)))!=0.0)
      return(CloseAll(lot));
//--- check for position type
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of closing the long position / delete pending orders
      if(CheckCloseLong())
        {
         DeleteOrders();
         return(true);
        }
     }
   else
     {
      //--- check the possibility of closing the short position / delete pending order
      if(CheckCloseShort())
        {
         DeleteOrders();
         return(true);
        }
     }
//--- return without operations
   return(false);
  }
```

CheckCloseLong

Checks conditions to close a long position.

```
virtual bool  CheckCloseLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks conditions to close a long position (CheckCloseLong() method of Signal object) and if they are satisfied, it closes the open position (CloseLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position close or limit/stop order delete         |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseLong()
  {
   double price=EMPTY_VALUE;
//--- check for long close operations
   if(m_signal.CheckCloseLong(price))
      return(CloseLong(price));
//--- return without operations
   return(false);
  }
```

CheckCloseShort

Checks conditions to close a short position.

```
virtual bool  CheckCloseShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks conditions to close a short position (CheckCloseShort() method of Signal object) and if they are satisfied, it closes the position (CloseShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position close or limit/stop order delete        |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseShort()
  {
   double price=EMPTY_VALUE;
//--- check for short close operations
   if(m_signal.CheckCloseShort(price))
      return(CloseShort(price));
//--- return without operations
   return(false);
  }
```

CloseAll

It performs partial of full position closing.

```
virtual bool  CloseAll(
   double     lot     // lot
   )
```

Parameters

lot

[in] Number of lots to reduce the position.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionClose method is used, which can also be used on accounts with the netting mode. The DeleteOrders() method is used to delete orders.

Implementation

```
//+------------------------------------------------------------------+
//| Position close and orders delete                                 |
//| INPUT:  lot - volume for close.                                  |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseAll(double lot)
  {
   bool result=false;
//--- check for close operations
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
      result=m_trade.PositionCloseByTicket(m_position.Identifier());
   else
     {
      if(m_position.PositionType()==POSITION_TYPE_BUY)
         result=m_trade.Sell(lot,0,0,0);
      else
         result=m_trade.Buy(lot,0,0,0);
     }
   result|=DeleteOrders();
//---
   return(result);
  }
```

Close

Closes the opened position.

```
virtual bool  Close()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

Closes the position (PositionClose() method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Position close                                                   |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Close()
  {
   return(m_trade.PositionClose(m_symbol.Name()));
  }
```

CloseLong

Closes the long position.

```
virtual bool  CloseLong(
   double    price     // price
   )
```

Parameters

price

[in] Market entry price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionCloseByTicket method is used.

Implementation

```
//+------------------------------------------------------------------+
//| Long position close                                              |
//| INPUT:  price - price for close.                                 |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseLong(double price)
  {
   bool result=false;
//---
   if(price==EMPTY_VALUE)
      return(false);
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
      result=m_trade.PositionCloseByTicket(m_position.Identifier());
   else
      result=m_trade.Sell(m_position.Volume(),price,0,0);
//---
   return(result);
  }
```

CloseShort

Closes the short position.

```
virtual bool  CloseShort(
   double     price     // price
   )
```

Parameters

price

[in] Market entry price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

In the "netting" mode, a position is closed using the CExpertTrade::Buy or CExpertTrade::Sell methods. In the "hedging" mode, the CTrade::PositionCloseByTicket method is used.

Implementation

```
//+------------------------------------------------------------------+
//| Short position close                                             |
//| INPUT:  price - price for close.                                 |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseShort(double price)
  {
   bool result=false;
//---
   if(price==EMPTY_VALUE)
      return(false);
   if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
      result=m_trade.PositionCloseByTicket(m_position.Identifier());
   else
      result=m_trade.Buy(m_position.Volume(),price,0,0);
//---
   return(result);
  }
```

CheckTrailingStop

It checks Trailing Stop conditions of the opened position.

```
virtual bool  CheckTrailingStop()
```

Return Value

true - a trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened position (CheckTrailingStopLong() or CheckTrailingStopShort() for long and short positions).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit position                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStop()
  {
//--- position must be selected before call
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of modifying the long position
      if(CheckTrailingStopLong()) return(true);
     }
   else
     {
      //--- check the possibility of modifying the short position
      if(CheckTrailingStopShort()) return(true);
     }
//--- return without operations
   return(false);
  }
```

CheckTrailingStopLong

It checks Trailing Stop conditions of the opened long position.

```
virtual bool  CheckTrailingStopLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened long position (CheckTrailingStopLong(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters (TrailingStopLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit long position                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopLong()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- check for long trailing stop operations
   if(m_trailing.CheckTrailingStopLong(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- long trailing stop operations
      return(TrailingStopLong(sl,tp));
     }
//--- return without operations
   return(false);
  }
```

CheckTrailingStopShort

It checks Trailing Stop conditions of the opened short position.

```
virtual bool  CheckTrailingStopShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions of the opened short position (CheckTrailingStopShort(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters (TrailingStopShort(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit short position                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopShort()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- check for short trailing stop operations
   if(m_trailing.CheckTrailingStopShort(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- short trailing stop operations
      return(TrailingStopShort(sl,tp));
     }
//--- return without operations
   return(false);
  }
```

TrailingStopLong

It modifies parameters of the opened long position.

```
virtual bool  TrailingStopLong(
   double     sl,     // Stop Loss
   double     tp,     // Take Profit
   )
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

The function modifies parameters of the opened long position (PositionModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing stop/profit long position                               |
//| INPUT:  sl - new stop loss,                                      |
//|         tp - new take profit.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopLong(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

TrailingStopShort

It modifies parameters of the opened short position.

```
virtual bool  TrailingStopLong(
   double     sl,     // Stop Loss
   double     tp,     // Take Profit
   )
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Return Value

true - trade operation has been executed, otherwise - false.

Note

The function modifies parameters of the opened short position (PositionModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing stop/profit short position                              |
//| INPUT:  sl - new stop loss,                                      |
//|         tp - new take profit.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopShort(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

CheckTrailingOrderLong

Checks Trailing Stop conditions of Buy Limit/Stop trailing order.

```
virtual bool  CheckTrailingOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions for buy limit/stop trailing order (CheckTrailingOrderLong() method of Trade Signals object) and modifies the order parameters if necessary (TrailingOrderLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing long limit/stop order                         |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderLong()
  {
   double price;
//--- check the possibility of modifying the long order
   if(m_signal.CheckTrailingOrderLong(GetPointer(m_order),price))
      return(TrailingOrderLong(m_order.PriceOpen()-price));
//--- return without operations
   return(false);
  }
```

CheckTrailingOrderShort

It checks Trailing Stop conditions of Sell Limit/Stop trailing order.

```
virtual bool  CheckTrailingOrderShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks Trailing Stop conditions for sell limit/stop trailing order (CheckTrailingOrderShort() method of Trade Signals object) and modifies the order parameters if necessary (TrailingOrderShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing short limit/stop order                        |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderShort()
  {
   double price;
//--- check the possibility of modifying the short order
   if(m_signal.CheckTrailingOrderShort(GetPointer(m_order),price))
      return(TrailingOrderShort(m_order.PriceOpen()-price));
//--- return without operations
   return(false);
  }
```

TrailingOrderLong

It modifies parameters of Buy Limit/Stop trailing order.

```
virtual bool  TrailingOrderLong(
   double     delta     // delta
   )
```

Parameters

delta

[in] Price delta.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It modifies parameters of Buy Limit/Stop trailing order (OrderModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing long limit/stop order                                   |
//| INPUT:  delta - price change.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderLong(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- modifying the long order
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
  }
```

TrailingOrderShort

It modifies parameters of Sell Limit/Stop trailing order.

```
virtual bool  TrailingOrderShort(
   double     delta     // delta
   )
```

Parameters

delta

[in] Price delta.

Return Value

true - trade operation has been executed, otherwise - false.

Note

It modifies parameters of Sell Limit/Stop trailing order (OrderModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing short limit/stop order                                  |
//| INPUT:  delta - price change.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderShort(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- modifying the short order
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
  }
```

CheckDeleteOrderLong

It checks conditions to delete Buy Limit/Stop order.

```
virtual bool  CheckDeleteOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It checks the order expiration time. It checks conditions to delete the Buy Limit/Stop order (CheckCloseLong(...) method of Signal class object) and deletes the order if condition is satisfied (DeleteOrderLong() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for delete long limit/stop order                           |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderLong()
  {
   double price;
//--- check the possibility of deleting the long order
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderLong());
     }
   if(m_signal.CheckCloseLong(price))
      return(DeleteOrderLong());
//--- return without operations
   return(false);
  }
```

CheckDeleteOrderShort

It checks conditions to delete Sell Limit/Stop order.

```
virtual bool  CheckDeleteOrderShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

1. It checks the order expiration time.

2. It checks conditions to delete the Sell Limit/Stop order (CheckCloseShort(...) method of Signal class object) and deletes the order if one of the conditions is satisfied (DeleteOrderShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for delete short limit/stop order                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderShort()
  {
   double price;
//--- check the possibility of deleting the short order
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderShort());
     }
   if(m_signal.CheckCloseShort(price))
      return(DeleteOrderShort());
//--- return without operations
   return(false);
  }
```

DeleteOrders

Deletes all orders.

```
virtual bool  DeleteOrders()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes all orders (DeleteOrder() for all orders).

Implementation

```
//+------------------------------------------------------------------+
//| Delete all limit/stop orders                                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrders()
  {
   bool result=false;
   int  total=OrdersTotal();
//---
   for(int i=total-1;i>=0;i--)
     {
      if(m_order.Select(OrderGetTicket(i)))
        {
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         result|=DeleteOrder();
        }
     }
//---
   return(result);
  }
```

DeleteOrder

Deletes the Limit/Stop order.

```
virtual bool  DeleteOrder()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes the Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete limit/stop order                                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrder()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

DeleteOrderLong

Deletes the Buy Limit/Stop order.

```
virtual bool  DeleteOrderLong()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes Buy Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete long limit/stop order                                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderLong()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

DeleteOrderShort

Deletes the Sell Limit/Stop order.

```
virtual bool  DeleteOrderShort()
```

Return Value

true - trade operation has been executed, otherwise - false.

Note

It deletes the Sell Limit/Stop order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete short limit/stop order                                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderShort()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

LotOpenLong

Gets trade volume for buy operation.

```
double  LotOpenLong(
   double     price,    // price
   double     sl        // Stop Loss
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for buy operation.

Note

It gets trade volume for buy operation (CheckOpenLong(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for open long position.               |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss.                                       |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotOpenLong(double price,double sl)
  {
   return(m_money.CheckOpenLong(price,sl));
  }
```

LotOpenShort

Gets trade volume for sell operation.

```
double  LotOpenShort(
   double     price,    // price
   double     sl        // Stop Loss
   )
```

Parameters

price

[in] Market entry price.

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for sell operation.

Note

It gets trade volume for sell operation (CheckOpenShort(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for open short position.               |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss.                                       |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotOpenShort(double price,double sl)
  {
   return(m_money.CheckOpenShort(price,sl));
  }
```

LotReverse

Gets trade volume for position reverse.

```
double  LotReverse(
   double     sl        // Stop Loss
   )
```

Parameters

sl

[in] Stop Loss price.

Return Value

Trade volume (in lots) for position reverse operation.

Note

It gets trade volume for position reverse operation (CheckReverse(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for reverse position.                  |
//| INPUT:  sl - stop loss.                                          |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotReverse(double sl)
  {
   return(m_money.CheckReverse(GetPointer(m_position),sl));
  }
```

PrepareHistoryDate

Sets starting date for tracking of trade history.

```
void  PrepareHistoryDate()
```

Note

The trade history tracking period is set from the beginning of the month (but not less than one day).

HistoryPoint

Creates a checkpoint of trade history.

```
void  HistoryPoint(
   bool    from_check_trade=false    // flag
   )
```

Parameters

from_check_trade=false

[in] Flag to avoid recursion.

Note

It saves the amount of positions, orders, deals, and historical orders.

CheckTradeState

Compares the current state with the saved one and calls the corresponding event handler.

```
bool  CheckTradeState()
```

Return Value

true - event has been handled, otherwise - false.

Note

It checks the number of positions, orders, deals, and historical orders by comparing with the values saved by HistoryPoint() method. If trade history has changed, it calls the corresponding virtual event handler.

WaitEvent

Sets the event waiting flag.

```
void  WaitEvent(
   ENUM_TRADE_EVENTS     event          // flag
   )
```

Parameters

event

[in] Flag of waiting for an event (from ENUM_TRADE_EVENTS enumeration) to set.

Return Value

None.

Event flags

```
//--- flags of expected events
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT                 =0,         // no expected events
   TRADE_EVENT_POSITION_OPEN            =0x1,       // flag of expecting the "opening of
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,       // flag of expecting of the "modifica
   TRADE_EVENT_POSITION_MODIFY          =0x4,       // flag of expecting of the "modifica
   TRADE_EVENT_POSITION_CLOSE           =0x8,       // flag of expecting of the "closing
   TRADE_EVENT_POSITION_STOP_TAKE       =0x10,      // flag of expecting of the "triggeri
   TRADE_EVENT_ORDER_PLACE              =0x20,      // flag of expecting of the "placing
   TRADE_EVENT_ORDER_MODIFY             =0x40,      // flag of expecting of the "modifica
   TRADE_EVENT_ORDER_DELETE             =0x80,      // flag of expecting of the "deletion
   TRADE_EVENT_ORDER_TRIGGER            =0x100      // flag of expecting of the "triggeri
  };
```

NoWaitEvent

Resets the event waiting flag.

```
void  NoWaitEvent(
   ENUM_TRADE_EVENTS     event          // flag
   )
```

Parameters

event

[in] Flag with events (from ENUM_TRADE_EVENTS enumeration) to reset.

Return Value

None.

Event flags

```
//--- flags of expected events
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT                 =0,         // no expected events
   TRADE_EVENT_POSITION_OPEN            =0x1,       // flag of expecting the "opening of
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,          // flag of expecting of the "modifica
   TRADE_EVENT_POSITION_MODIFY          =0x4,       // flag of expecting of the "modifica
   TRADE_EVENT_POSITION_CLOSE           =0x8,       // flag of expecting of the "closing
   TRADE_EVENT_POSITION_STOP_TAKE       =0x10,      // flag of expecting of the "triggeri
   TRADE_EVENT_ORDER_PLACE              =0x20,      // flag of expecting of the "placing
   TRADE_EVENT_ORDER_MODIFY             =0x40,      // flag of expecting of the "modifica
   TRADE_EVENT_ORDER_DELETE             =0x80,      // flag of expecting of the "deletion
   TRADE_EVENT_ORDER_TRIGGER            =0x100      // flag of expecting of the "triggeri
  };
```

TradeEventPositionStopTake

Event handler of the "Position Stop Loss/Take Profit triggered" event.

```
virtual bool  TradeEventPositionStopTake()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventOrderTriggered

Event handler of the "Pending Order triggered" event.

```
virtual bool  TradeEventOrderTriggered()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventPositionOpened

Event handler of the "Position opened" event.

```
virtual bool  TradeEventPositionOpened()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventPositionVolumeChanged

Event handler of the "Position volume changed" event.

```
virtual bool  TradeEventPositionVolumeChanged()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventPositionModified

Event handler of the "Position modified" event.

```
virtual bool  TradeEventPositionModified()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventPositionClosed

Event handler of the "Position closed" event.

```
virtual bool  TradeEventPositionClosed()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventOrderPlaced

Event handler of the "Pending order placed" event.

```
virtual bool  TradeEventOrderPlaced()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventOrderModified

Event handler of the "Pending order modified" event.

```
virtual bool  TradeEventOrderModified()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventOrderDeleted

Event handler of the "Pending order deleted" event.

```
virtual bool  TradeEventOrderDeleted()
```

Return Value

The CExpert class method does nothing and always returns true.

TradeEventNotIdentified

Event handler of the non-identified event.

```
virtual bool  TradeEventNotIdentified()
```

Return Value

The CExpert class method does nothing and always returns true.

Note

Note that several trade events can arrive, in such cases it is difficult to identify them.

TimeframeAdd

Add a timeframe for tracking.

```
void  TimeframeAdd(
   ENUM_TIMEFRAMES     period          // timeframe
   )
```

Parameters

period

[in] Timeframe (from ENUM_TIMEFRAMES enumeration) to be tracked.

Return Value

None.

TimeframesFlags

Forms the timeframe flags.

```
int  TimeframesFlags(
   MqlDateTime&     time          // reference
   )
```

Parameters

time

[in] Reference to MqlDateTime type structure containing a new time.

Return Value

It returns the flag that indicates timeframes with a new bar.

CExpertSignal

CExpertSignal is a base class for trading signals, it does nothing (except CheckReverseLong() and CheckReverseShort() methods) but provides the interfaces.

How to use it:

1. Prepare an algorithm for trading signals;
2. Create your own trading signal class, inherited from CExpertSignal class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trading signal classes in the Expert\Signal\ folder.

Description

CExpertSignal is a base class for implementation of trading signal algorithms.

Declaration

```
   class CExpertSignal : public CExpertBase
```

Title

```
   #include <Expert\ExpertSignal.mqh>
```

Inheritance hierarchy

CObject
  CExpertBase
    CExpertSignal

Direct descendants

CSignalAC, CSignalAMA, CSignalAO, CSignalBearsPower, CSignalBullsPower, CSignalCCI, CSignalDeM, CSignalDEMA, CSignalEnvelopes, CSignalFrAMA, CSignalRSI, CSignalRVI, CSignalSAR, CSignalStoch, CSignalTEMA, CSignalTriX, CSignalWPR

Class Methods by Groups

Initialization	
virtual InitIndicators	Initializes indicators and timeseries
virtual ValidationSettings	Checks the object settings
virtual AddFilter	Adds a filter to combined signal
Access to Protected Data	
BasePrice	Sets base price level
UsedSeries	Gets the flags of timeseries used
Parameters Setting	

Initialization	
Weight	Sets the value of "Weight" parameter
PatternsUsage	Sets the value of "PatternsUsage" parameter
General	Sets the value of "General" parameter
Ignore	Sets the value of "Ignore" parameter
Invert	Sets the value of "Invert" parameter
ThresholdOpen	Sets the value of "ThresholdOpen" parameter
ThresholdClose	Sets the value of "ThresholdClose" parameter
PriceLevel	Sets the value of "PriceLevel" parameter
StopLevel	Sets the value of "StopLevel" parameter
TakeLevel	Sets the value of "TakeLevel" parameter
Expiration	Sets the value of "Expiration" parameter
Magic	Sets the value of "Magic" parameter
Checking Trading Conditions	
virtual CheckOpenLong	Checks conditions to open long position
virtual CheckCloseLong	Checks conditions to close long position
virtual CheckOpenShort	Checks conditions to open short position
virtual CheckCloseShort	Checks conditions to close short position
virtual CheckReverseLong	Checks conditions of long position reversal
virtual CheckReverseShort	Checks conditions of short position reversal
Trade Parameters Setting	
virtual OpenLongParams	Sets parameters for long position opening
virtual OpenShortParams	Sets parameters for short position opening
virtual CloseLongParams	Sets parameters for long position closing
virtual CloseShortParams	Sets parameters for short position closing
Checking of Order Trailing Conditions	
virtual CheckTrailingOrderLong	Checks conditions to modify parameters of Buy Pending order
virtual CheckTrailingOrderShort	Checks conditions to modify parameters of Sell Pending order

Initialization	
Methods to Check Formation of Market Orders	
virtual LongCondition	Gets the result of checking buy conditions
virtual ShortCondition	Gets the result of checking sell conditions
virtual Direction	Gets the "weighted" direction of price

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

BasePrice

Sets base price level.

```
void  BasePrice(
   double     value          // value
   )
```

Parameters

value

[in] New value of the parameter.

Return Value

None.

UsedSeries

Gets the flags of the timeseries used.

```
int  UsedSeries()
```

Return Value

Flags of the used timeseries (if the symbol/timeframe corresponds to the working symbol/timeframe), otherwise 0.

Weight

Sets new value of "Weight" parameter.

```
void  Weight(
   double     value          // value
   )
```

Parameters

value

[in] "Weight" parameter.

Return Value

None.

PatternUsage

Sets new value of "PatternsUsage" parameter.

```
void  PatternUsage(
   double     value          // value
   )
```

Parameters

value

[in] New value of "PatternsUsage".

Return Value

None.

General

Sets new value of "General" parameter.

```
void  General(
   int     value          // value
   )
```

Parameters

value

[in] New value of "General".

Return Value

None.

Ignore

Sets new value of "Ignore" parameter.

```
void  Ignore(
   long     value          // value
   )
```

Parameters

value

[in] New value of "Ignore".

Return Value

None.

Invert

Sets new value of "Invert" parameter.

```
void  Invert(
   long     value          // value
   )
```

Parameters

value

[in] New value of "Invert".

Return Value

None.

ThresholdOpen

Sets new value of "ThresholdOpen" parameter.

```
void  ThresholdOpen(
   long     value          // value
   )
```

Parameters

value

[in] New value of "ThresholdOpen".

Return Value

None.

Note

The range of "ThresholdOpen" parameter is from 0 to 100. Used when "voting" to open position.

ThresholdClose

Sets new value of "ThresholdClose" parameter.

```
void  ThresholdOpen(
   long     value          // value
   )
```

Parameters

value

[in] New value of "ThresholdClose".

Return Value

None.

Note

The range of "ThresholdClose" parameter is from 0 to 100. Used when "voting" to close position.

PriceLevel

Sets new value of "PriceLevel" parameter.

```
void  PriceLevel(
   double    value          // value
   )
```

Parameters

value

[in] New value of "PriceLevel".

Return Value

None.

Note

The value of "PriceLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "PriceLevel" is used to define the open price relative to the base price.

StopLevel

Sets new value of "StopLevel" parameter.

```
void  StopLevel(
   double     value          // value
   )
```

Parameters

value

[in] New value of "StopLevel".

Return Value

None.

Note

The value of "StopLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "StopLevel" is used to define the Stop Loss price relative to the open price.

TakeLevel

Sets new value of "TakeLevel" parameter.

```
void  TakeLevel(
   double     value          // value
   )
```

Parameters

value

[in] New value of "TakeLevel".

Return Value

None.

Note

The value of "TakeLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "TakeLevel" is used to define the Take Profit price relative to the open price.

Expiration

Sets the value of "Expiration" parameter.

```
void  Expiration(
   int     value          // value
   )
```

Parameters

value

[in] New value of "Expiration".

Return Value

None.

Note

The value of "Expiration" parameter is defined in bars. It is used as Expiration time for Pending Orders (when trading using pending orders).

Magic

Sets the value of "Magic" parameter.

```
void  Magic(
   int     value          // value
   )
```

Parameters

value

[in] New value of "Magic" (Expert Advisor ID).

Return Value

None.

ValidationSettings

Checks the object settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - object settings are correct, otherwise - false.

InitIndicators

Initializes all necessary indicators and timeseries.

```
virtual bool  InitIndicators(
   CIndicators*   indicators     // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Return Value

true - successful completion, otherwise - false.

Note

The necessary timeseries are initialized only if the object uses the symbol or timeframe different from the one defined at initialization.

AddFilter

Adds a filter to the composite signal.

```
virtual bool  AddFilter(
   CExpertSignal*   filter     // pointer
   )
```

Parameters

indicators

[in] Pointer to filter object.

Return Value

true - successful, otherwise - false.

CheckOpenLong

Checks conditions to open a long position.

```
virtual bool  CheckOpenLong(
   double&     price,          // price
   double&     sl,             // Stop Loss
   double&     tp,             // Take Profit
   datetime&   expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckOpenShort

Checks conditions to open a short position.

```
virtual bool  CheckOpenShort(
   double&     price,          // price
   double&     sl,             // Stop Loss
   double&     tp,             // Take Profit
   datetime&   expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

OpenLongParams

Sets parameters to open a long position.

```
virtual bool  OpenLongParams(
   double&     price,          // price
   double&     sl,             // Stop Loss
   double&     tp,             // Take Profit
   datetime&   expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for open price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - successful, otherwise - false.

OpenShortParams

Sets parameters to open a short position.

```
virtual bool  OpenShortParams(
   double&     price,          // price
   double&     sl,             // Stop Loss
   double&     tp,             // Take Profit
   datetime&   expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - successful, otherwise - false.

CheckCloseLong

Checks conditions to close a long position.

```
virtual bool  CheckCloseLong(
   double&  price       // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

CheckCloseShort

Checks conditions to close a short position.

```
virtual bool  CheckCloseShort(
   double&  price       // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

CloseLongParams

Sets parameters to close a long position.

```
virtual bool  CloseLongParams(
   double&    price           // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - successful, otherwise - false.

CloseShortParams

Sets parameters to close a short position.

```
virtual bool  CloseShortParams(
   double&     price          // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Return Value

true - successful, otherwise - false.

CheckReverseLong

Checks conditions of a long position reversal.

```
virtual bool  CheckReverseLong(
   double&     price,           // price
   double&     sl,              // Stop Loss
   double&     tp,              // Take Profit
   datetime&   expiration       // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckReverseShort

Checks conditions of a short position reversal.

```
virtual bool  CheckReverseShort(
   double&     price,          // price
   double&     sl,             // Stop Loss
   double&     tp,             // Take Profit
   datetime&   expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for reversal price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference (if necessary).

Return Value

true - condition is satisfied, otherwise - false.

CheckTrailingOrderLong

Checks conditions to modify parameters of Buy Pending order.

```
virtual bool  CheckTrailingOrderLong(
   COrderInfo*     order,           // order
   double&         price            // price
   )
```

Parameters

order

[in] Pointer to COrderInfo class object.

price

[in][out] Variable for Stop Loss price.

Return Value

true - condition is satisfied, otherwise - false.

CheckTrailingOrderShort

Checks conditions to modify parameters of Sell Pending order.

```
virtual bool  CheckTrailingOrderShort(
   COrderInfo*     order,           // order
   double&         price            // price
   )
```

Parameters

order

[in] Pointer to COrderInfo class object.

price

[in][out] Variable for Stop Loss price.

Return Value

true - condition is satisfied, otherwise - false.

LongCondition

Checks conditions to open a long position.

```
virtual int  LongCondition()
```

Return Value

If the conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal). If there is no signal to open a long position, it returns 0.

Note

The base class has no implementation of checking conditions to open a long position and always returns 0.

ShortCondition

Checks conditions to open a short position.

```
virtual int  ShortCondition()
```

Return Value

If the conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal). If there isn't a signal to open short position, it returns 0.

Note

The base class has no implementation of checking conditions to open a short position and always returns 0.

Direction

Returns the value of "weighted" price direction.

```
virtual double  Direction()
```

Return Value

It returns the value>0 when upward direction is most probable and <0 in case of a downward direction. The absolute value depends on the "strength" of a signal.

Note

If the built-in filters are used, their results are considered when defining the general direction.

CExpertTrailing

CExpertTrailing is a base class for trailing algorithms, it does nothing but provides the interfaces.

How to use it:

1. Prepare an algorithm for trailing;
2. Create your own trailing class inherited from CExpertTrailing class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trailing classes in the Expert\Trailing\ folder.

Description

CExpertTrailing is a base class for implementation of trailing stop algorithms.

Declaration

```
   class CExpertTrailing : public CExpertBase
```

Title

```
   #include <Expert\ExpertTrailing.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertTrailing

Direct descendants

CTrailingFixedPips, CTrailingMA, CTrailingNone, CTrailingPSAR

Class Methods by Groups

Checking of Trailing Stop Conditions	
virtual CheckTrailingStopLong	Checks conditions to modify parameters of the long position
virtual CheckTrailingStopShort	Checks conditions to modify parameters of the short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries, InitIndicators

CheckTrailingStopLong

Checks conditions to modify parameters of a long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // pointer
   double&         sl,            // Stop Loss
   double&         tp             // Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

Note

Base class method always returns false.

CheckTrailingStopShort

Checks conditions to modify parameters of a short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,      // pointer
   double&         sl,            // Stop Loss
   double&         tp             // Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Return Value

true - condition is satisfied, otherwise - false.

Note

Base class method always returns false.

CExpertMoney

CExpertMoney is a base class for money and risk management algorithms.

Description

CExpertMoney is a base class for implementation of money and risk management classes.

Declaration

```
   class CExpertMoney : public CObject
```

Title

```
   #include <Expert\ExpertMoney.mqh>
```

Inheritance hierarchy

CObject
  CExpertBase
    CExpertMoney

Direct descendants

CMoneyFixedLot, CMoneyFixedMargin, CMoneyFixedRisk, CMoneyNone, CMoneySizeOptimized

Class Methods by Groups

Access to Protected Data	
Percent	Sets the value of "Risk percent" parameter
Initialization	
virtual ValidationSettings	Checks the settings
Checking Trading Conditions	
virtual CheckOpenLong	Gets the volume for a long position
virtual CheckOpenShort	Gets the volume for a short position
virtual CheckReverse	Gets the volume for a reverse of the position
virtual CheckClose	Checks conditions to close an opened position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Percent

Sets the value of "Risk percent" parameter.

```
void  Percent(
   double  percent      // risk percent
   )
```

Parameters

percent

[in] Risk percent.

Return Value

None.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

Base class method always returns true.

CheckOpenLong

Gets the volume for a long position.

```
virtual double  CheckOpenLong(
   double  price,      // price
   double  sl          // Stop Loss
   )
```

Parameters

price

[in] Opening price of a long position.

sl

[in] Stop Loss price of a long position.

Return Value

Trade volume for a long position.

CheckOpenShort

Gets the volume for a short position.

```
virtual double  CheckOpenShort(
   double  price,      // price
   double  sl          // Stop Loss
   )
```

Parameters

price

[in] Opening price for a short position.

sl

[in] Stop Loss price of a short position.

Return Value

Trade volume for a short position.

CheckReverse

Gets the volume for reverse of the position.

```
virtual double  CheckReverse(
   CPositionInfo*   position,      // pointer
   double           sl             // Stop Loss
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in] Stop Loss price.

Return Value

Volume for reverse of the position.

CheckClose

Checks conditions to close the opened position.

```
virtual double  CheckClose()
```

Return Value

true - condition is satisfied, otherwise - false.

Modules of Trade Signals

The standard delivery of the client terminal includes a set of ready-made modules of trade signals for "MQL5 Wizard". When creating an Expert Advisor in MQL5 Wizard, you can use any combination of the modules of trade signals (up to 64). The final decision on a trade operation is made on the basis of complex analysis of signals obtained from all included modules. The detailed description of the mechanism of making trade decisions is given below.

The standard delivery includes the following modules of signals:

- Signals of the Indicator Accelerator Oscillator
- Signals of the Indicator Adaptive Moving Average
- Signals of the Indicator Awesome Oscillator
- Signals of the Oscillator Bears Power
- Signals of the Oscillator Bulls Power
- Signals of the Oscillator Commodity Channel Index
- Signals of the Oscillator DeMarker
- Signals of the Indicator Double Exponential Moving Average
- Signals of the Indicator Envelopes
- Signals of the Indicator Fractal Adaptive Moving Average
- Signals of the Intraday Time Filter
- Signals of the Oscillator MACD
- Signals of the Indicator Moving Average
- Signals of the Indicator Parabolic SAR
- Signals of the Oscillator Relative Strength Index
- Signals of the Oscillator Relative Vigor Index
- Signals of the Oscillator Stochastic
- Signals of the Oscillator Triple Exponential Average
- Signals of the Indicator Triple Exponential Moving Average
- Signals of the Oscillator Williams Percent Range

The Mechanism of Making Trade Decisions on the Basis of Signal Modules

The mechanism of making trade decisions can be represented as the following list of basic principles:

- Each of the modules of signals has its set of market modules (certain combination of prices and values of an indicator).
- Each market model has a significance that may vary with the range of 1 to 100. The higher is the significance, the stronger the model is.
- Each of the models generates a forecast of direction of the price movement.
- A forecast of a module is the result of search for embedded models, and it is outputted as a number within the range of -100 to 100. The sign determines the direction of forecast movement (negative

sign means the price will fall, positive sign means the price will rise). The absolute value corresponds to the strength of the best found model.

- The forecast of each module is sent to the final "voting" with a weight coefficient of 0 to 1 specified in its settings ("Weight").
- The result of voting is a number within the range of -100 to 100, where the sign determines direction of the forecast movement, and the absolute value characterizes the strength of the signal. It is calculated as the Arithmetic mean of weighted forecasts of all the modules of signals. The absolute value is used by an Expert Advisor to make trade decisions.

Each generated Expert Advisor has two adjustable settings — threshold levels of opening and closing a position (ThresholdOpen and ThresholdClose) that can be equal to a value in the range of 0 to 100. If the strength of final signal exceeds a threshold level, a trade operation that corresponds to the sign of the signal is performed.

Examples

Consider an Expert Advisor with the following threshold levels: ThresholdOpen=20 and ThresholdClose=90. Two modules of signals participate in making decisions on trade operations: the MA module with weight 0.4 and the Stochastic module with weight 0.8. Let's analyze two variants of obtained trade signals:

Variant 1

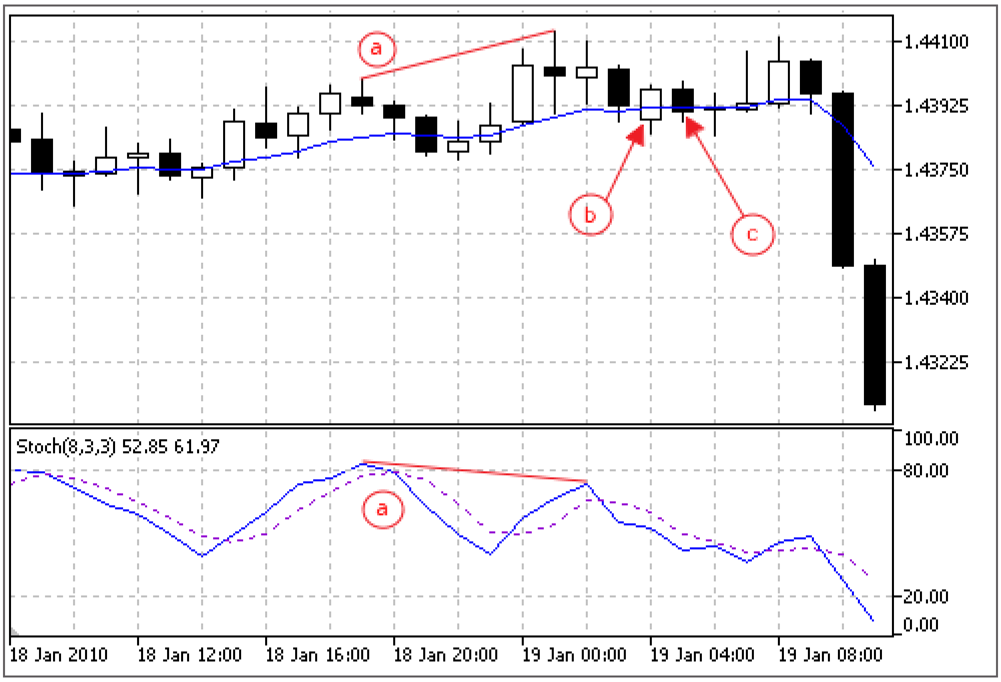
The price crossed the rising MA upwards. This case corresponds to one of the market models implemented in the MA module. This model implies a rise of price. Its significance is equal to 100. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the Stochastic module. This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as 0.4 * 100 = 40. The value from the Stochastic module is calculated as 0.8 * (-80) = -64. The final value is calculated as the Arithmetic mean of these two rates: (40 - 64)/2 = -12. The result of voting is the signal for selling with relative strength equal to 12. The threshold level that is equal to 20 is not reached. Thus a trade operation is not performed.

Variant 2

The price crossed the rising MA downwards. This case corresponds to one of the models implemented in the MA module.This model implies a rise of price. Its significance is equal to 10. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the Stochastic module. This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as 0.4 * 10 = 4. The value from the Stochastic module is calculated as 0.8 * (-80) = -64. The final value is calculated as the Arithmetic mean of these two rates: (4 - 64)/2 = -30. The result of voting is the signal for selling with relative strength equal to 30. The threshold level that is equal to 20 is reached. Thus the result is the signal for opening a short position.



a) Divergence of the price and the Stochastic oscillator (variants 1 and 2).

b) The price crossed the MA indicator upwards (variant 1).

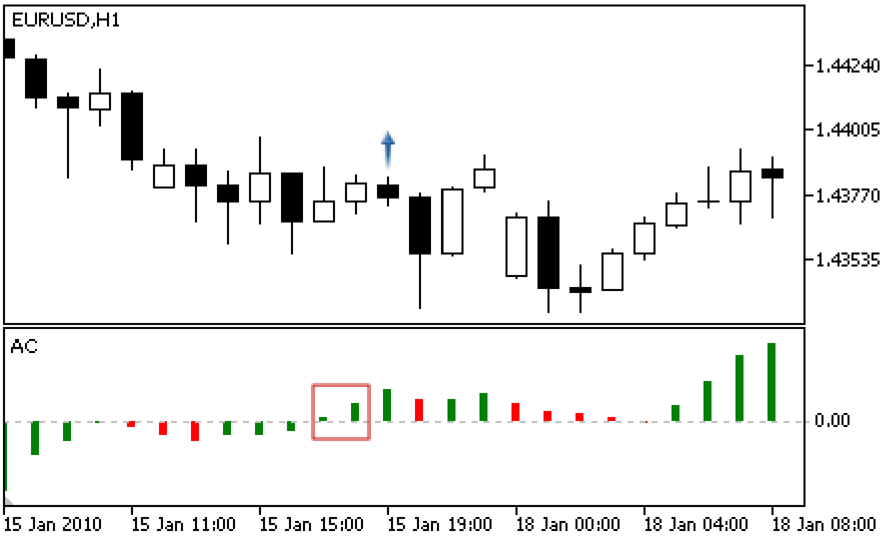
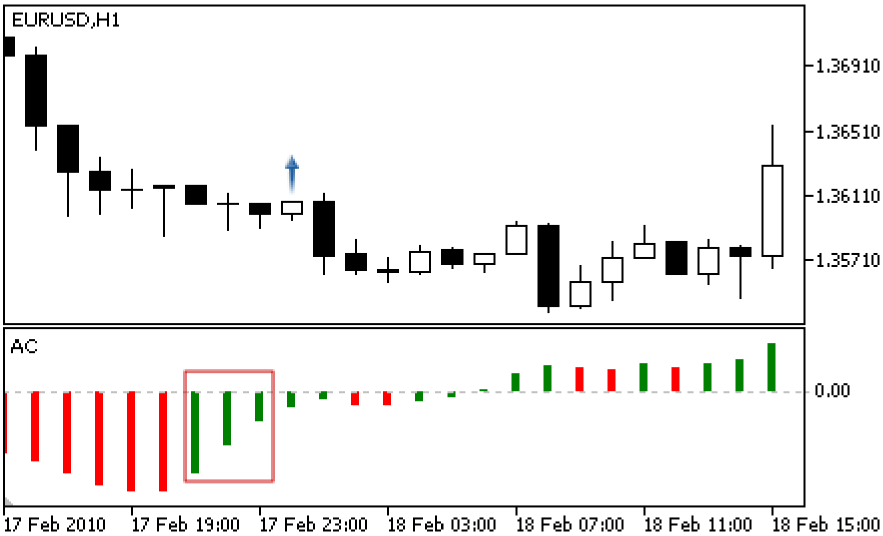
c) The price crossed the MA indicator downwards (variant 2).

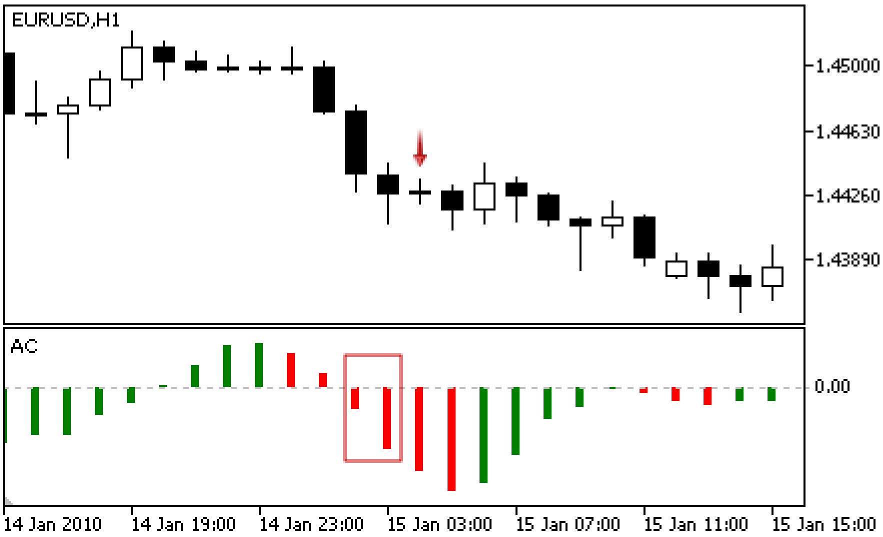
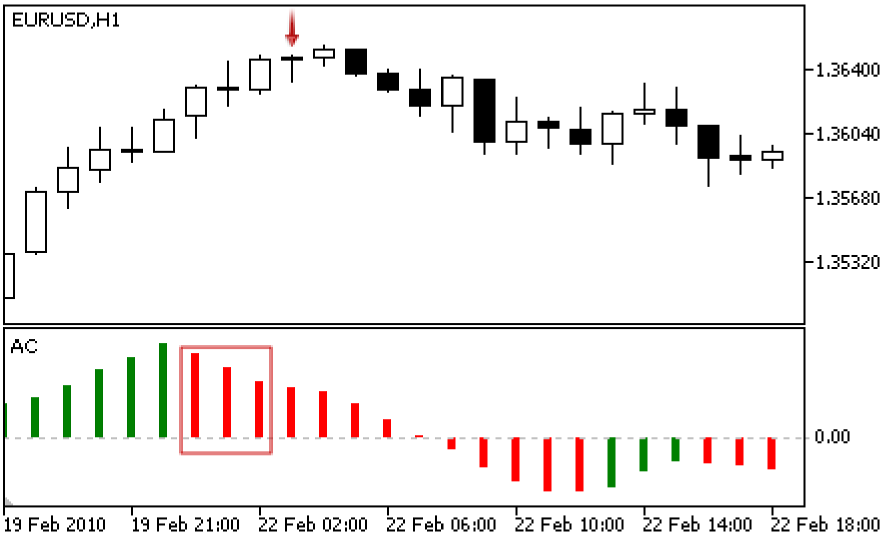
Signals of the Indicator Accelerator Oscillator

This module is based on the market models of the indicator Accelerator Oscillator. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- The indicator value is above 0 and it rises at the analyzed and at the previous bars.  - The indicator value is below 0 and it rises at the analyzed and at the previous bars. 
For selling	- The indicator value is below 0 and it falls at the analyzed and at the previous bars.

Signal Type	Description of Conditions
	 - The indicator value is above 0 and it falls at the analyzed and at the two previous bars. 
No objections to buying	The indicator value grows at the analyzed bar.
No objections to selling	The indicator value falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only"), an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

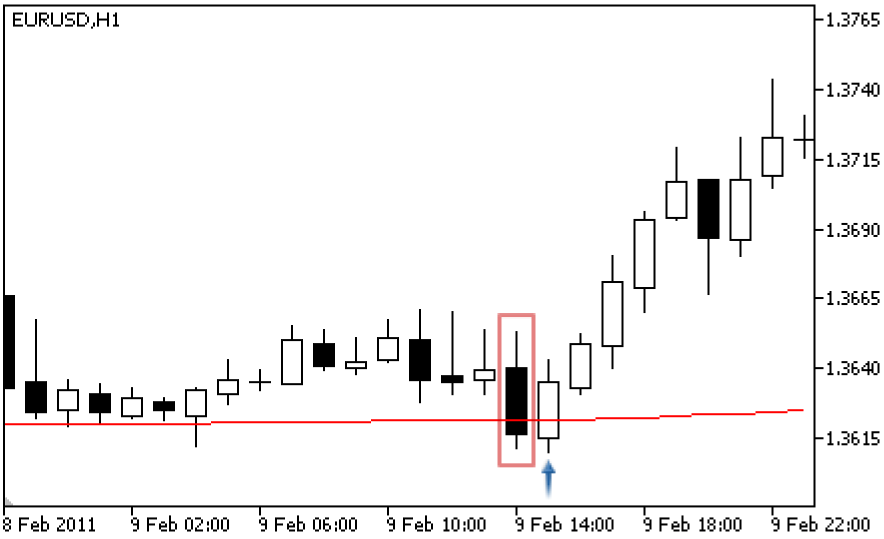
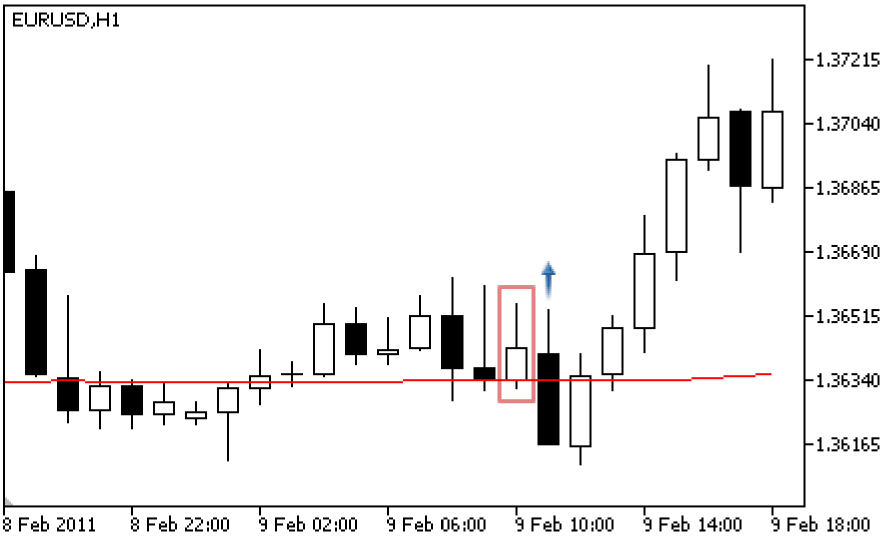
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

Signals of the Indicator Adaptive Moving Average

This module is based on the market models of the indicator Adaptive Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

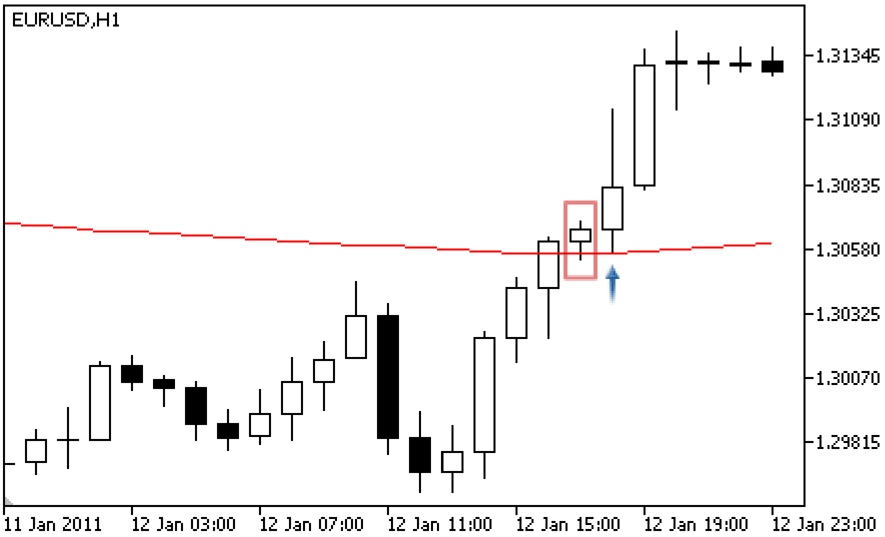
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- **Failed breakout.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) but the indicator rises (weak indicator line roll-back signal).  - **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

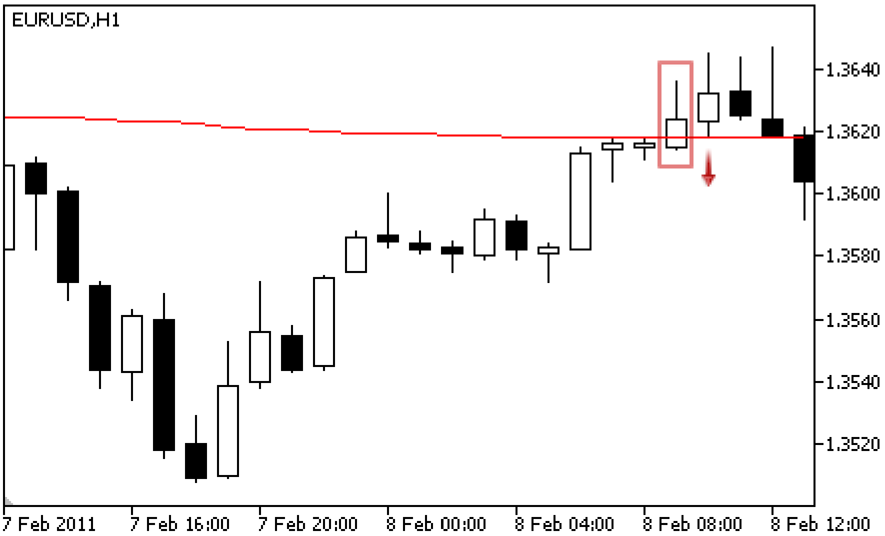
<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>

- **True breakout.** The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (indicator line roll-back signal).



</td></tr>
<tr><td>For selling</td><td>

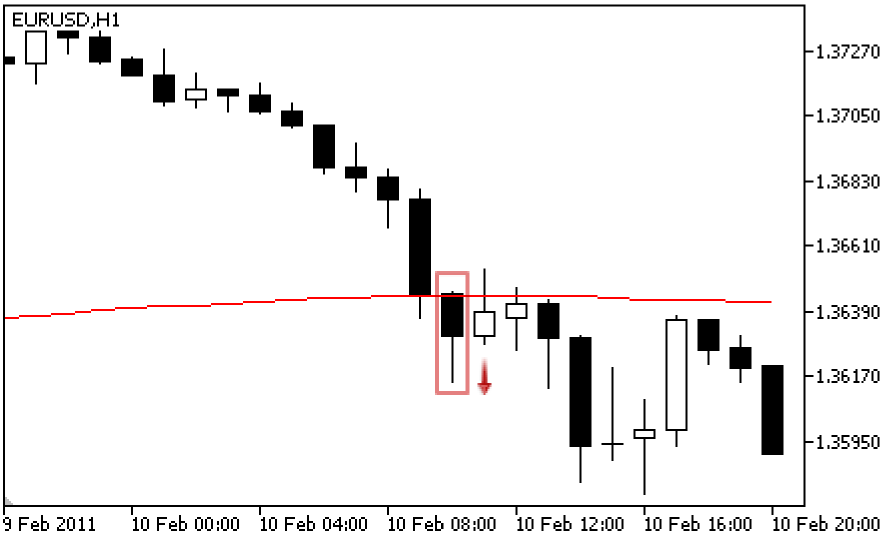
- **Failed breakout.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) but the indicator falls (weak indicator line roll-back signal).



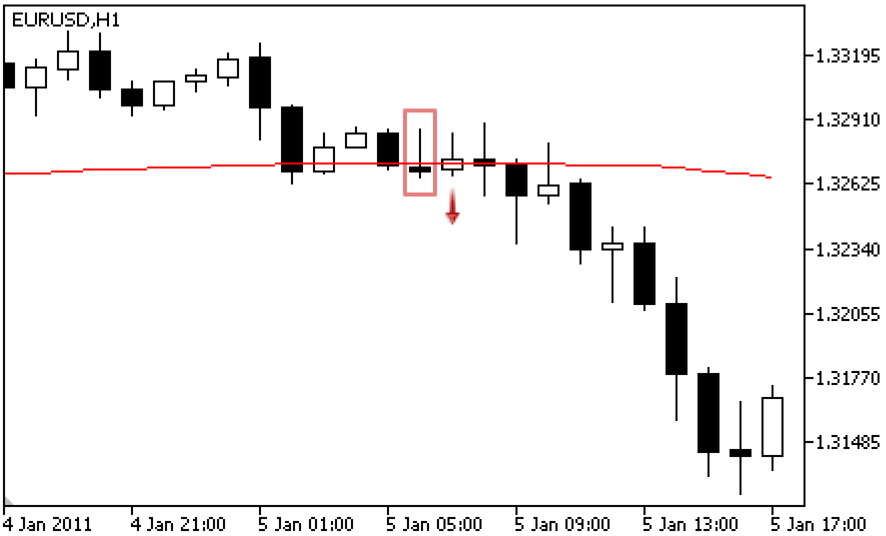
- **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

</td></tr>
</table>

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>



- **True breakout.** The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (indicator line roll-back signal).



</td></tr>
<tr><td>No objections to buying</td><td>The price is above the indicator.</td></tr>
<tr><td>No objections to selling</td><td>The price is below the indicator.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only"), an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Indicator Awesome Oscillator

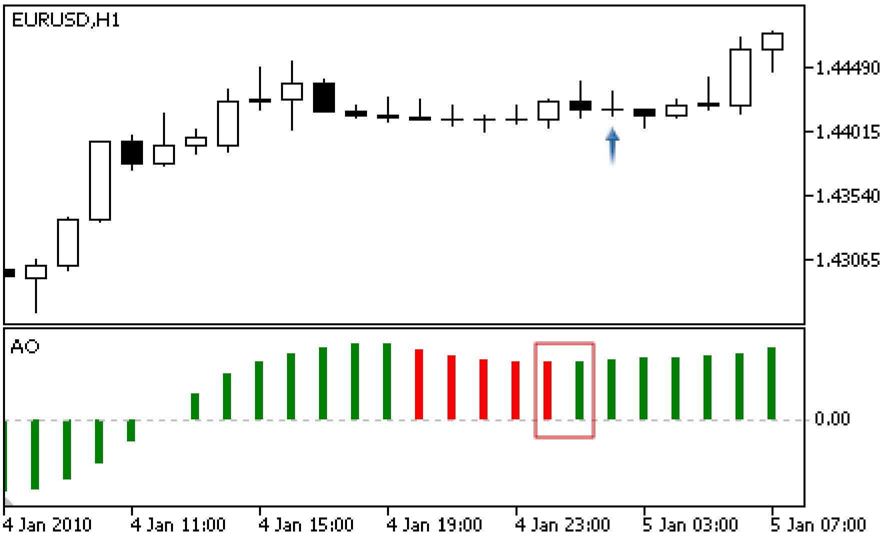
This module of signals is based on the market models of the indicator Awesome Oscillator. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

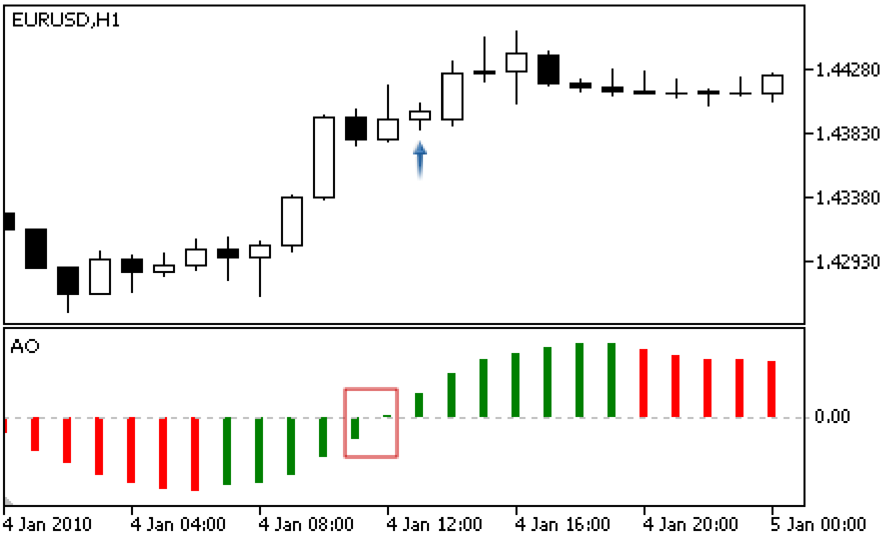
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Saucer** — value of the indicator at the analyzed bar rises, and it fell at the previous bars; at that, both values are above 0.

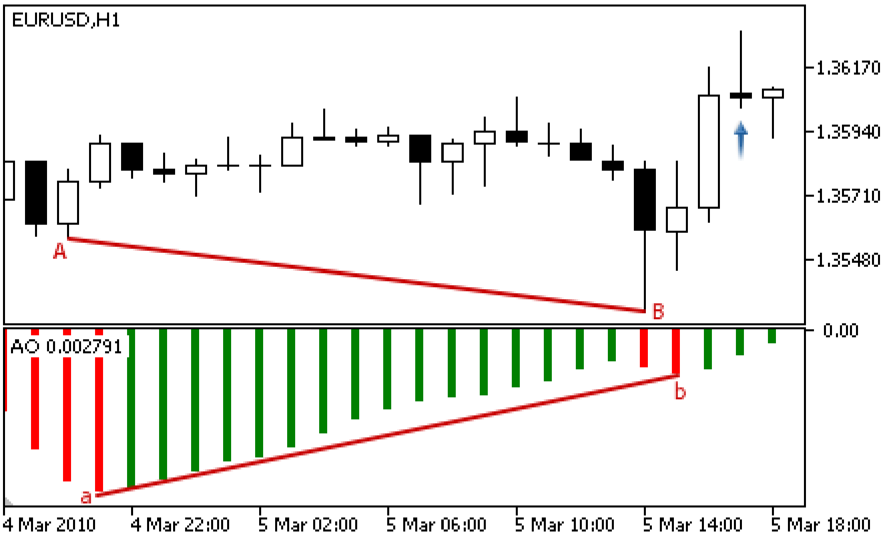
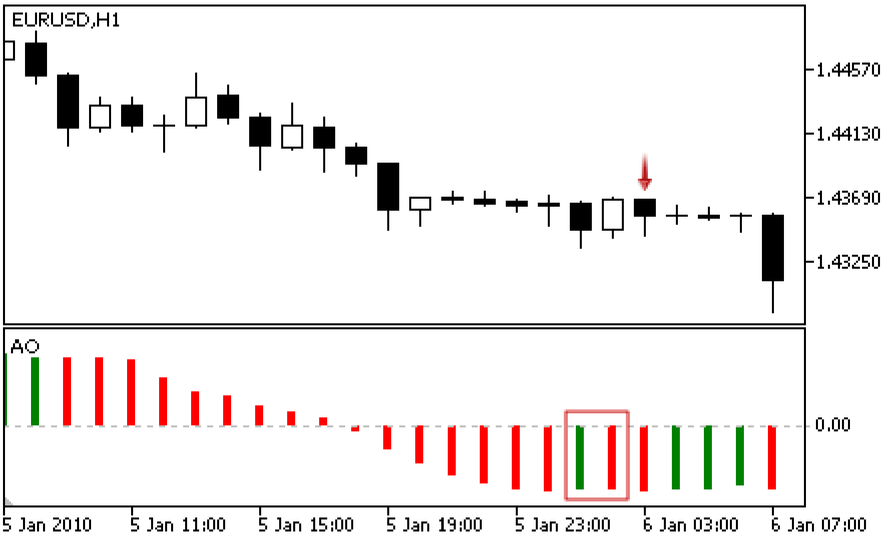


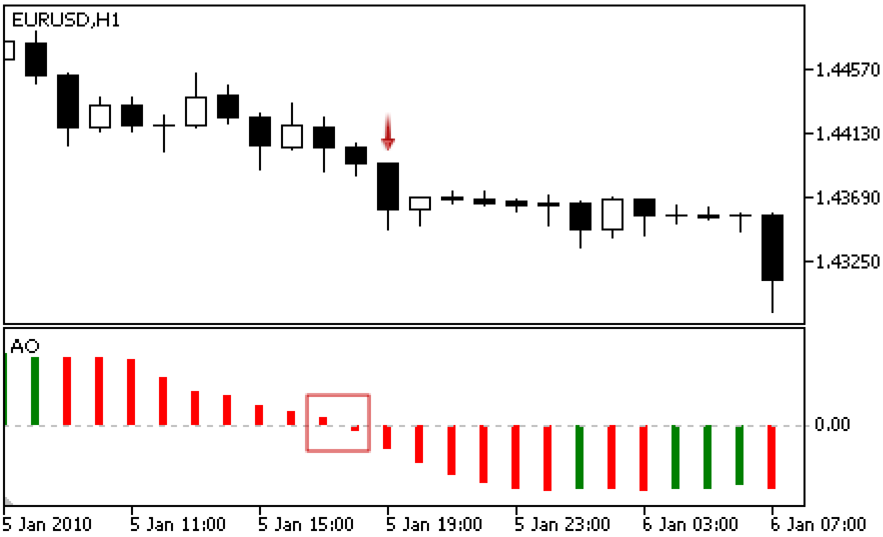
- **Crossing the zero line** — value of the indicator is above 0 at the analyzed bar, and it is below 0 at the previous bar.



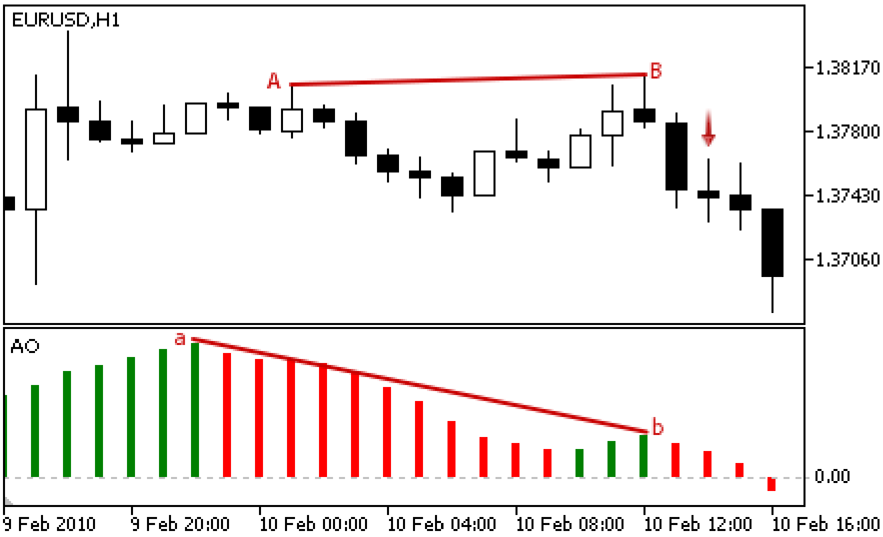
- **Divergence** — the first analyzed bottom of the indicator is shallower than the previous one, and the corresponding price bottom is deeper

</td></tr>
</table>

Signal Type	Description of Conditions
	than the previous one. In addition, the indicator must not rise above the zero level. 
For selling	- **Saucer** — value of the indicator at the analyzed bar falls, and it rose at the previous bars; at that, both values are below 0.  - **Crossing the zero line** — value of the indicator is below 0 at the analyzed bar, and it is above 0 at the previous bar.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>


- **Divergence** — the first analyzed peak of the indicator is lower than the previous one, and the corresponding price peak is higher than the previous one. In addition, the indicator must not falls below the zero level.



</td></tr>
<tr><td>No objections to buying</td><td>The indicator value grows at the analyzed bar.</td></tr>
<tr><td>No objections to selling</td><td>The indicator value falls at the analyzed bar.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

Signals of the Oscillator Bears Power

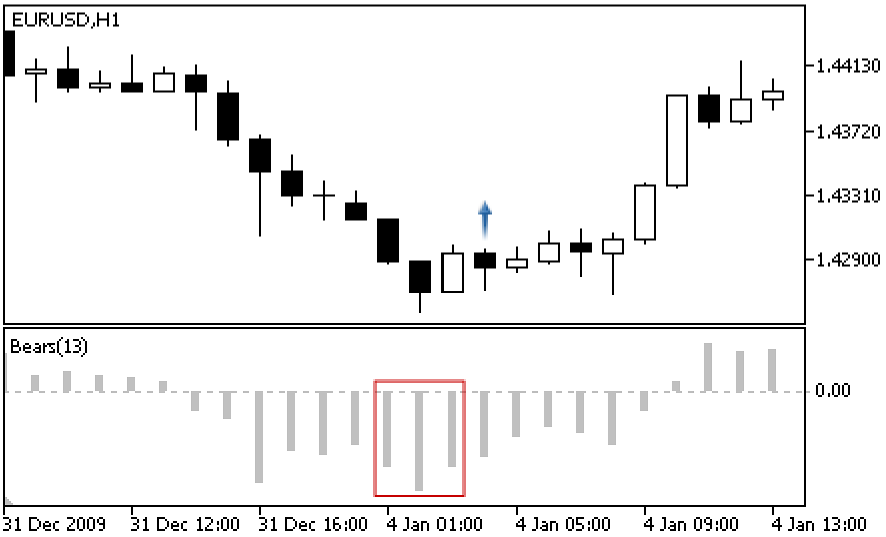
This module of signals is based on the market models of the oscillator Bears Power. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

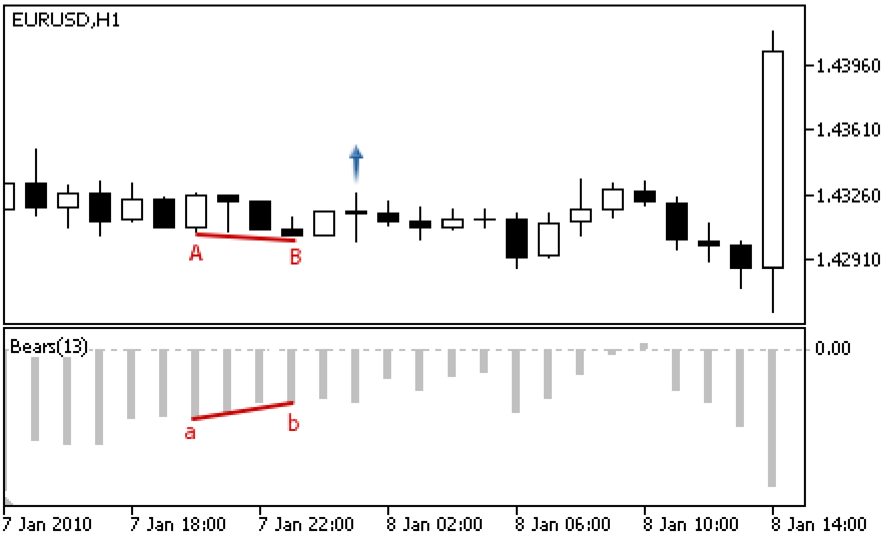
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Reverse** — the oscillator turned upwards and its value at the analyzed bar is below 0.



- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. In addition, the oscillator must not rise above the zero level.



</td></tr>
<tr><td>For selling</td><td>No signals for selling.</td></tr>
</table>

Signal Type	Description of Conditions
No objections to buying	Value of the oscillator is less than 0.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBears	Period of calculation of the oscillator.

Signals of the Oscillator Bulls Power

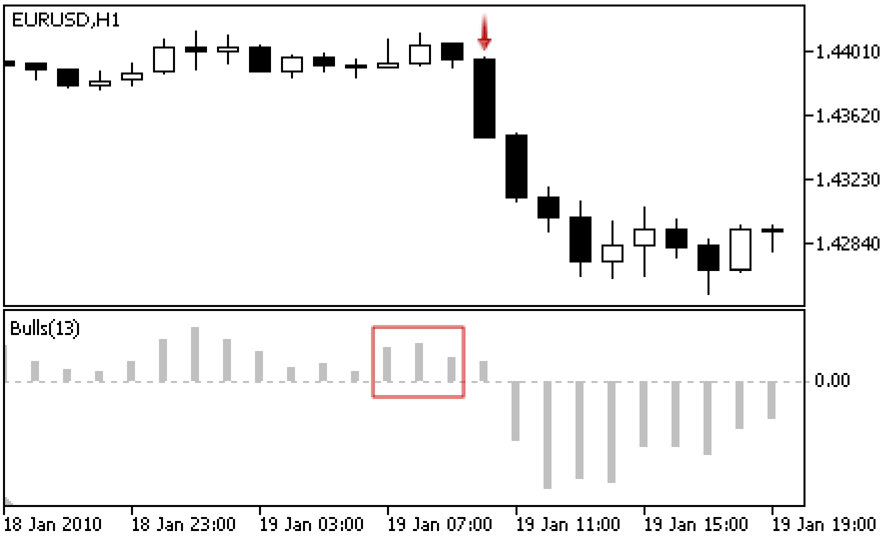
This module of signals is based on the market models of the oscillator Bulls Power. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

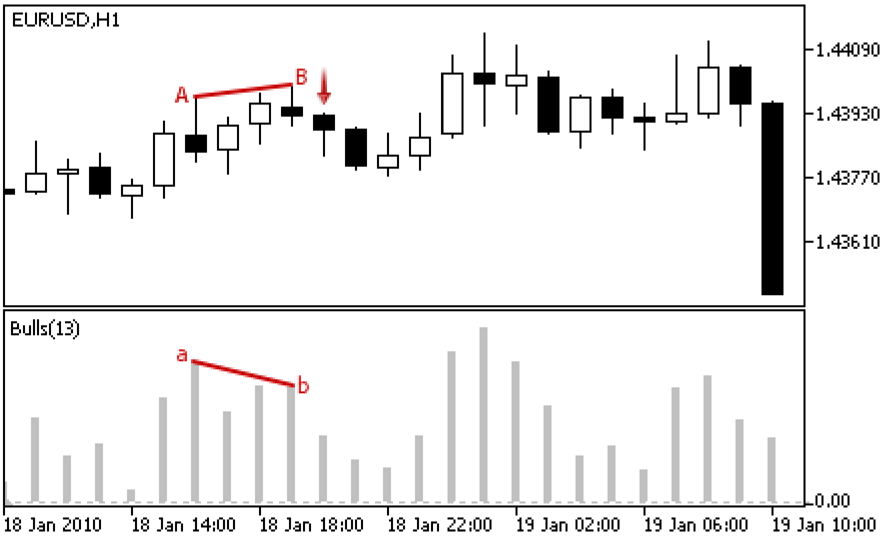
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>No signals for buying.</td></tr>
<tr><td>For selling</td><td>

- **Reverse** — the oscillator turned downwards and its value at the analyzed bar is above 0.



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. In addition, the oscillator must not fall below the zero level.



</td></tr>
</table>

Signal Type	Description of Conditions
No objections to buying	No signals.
No objections to selling	Value of the oscillator is greater than 0.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBulls	Period of calculation of the oscillator.

Signals of the Oscillator Commodity Channel Index

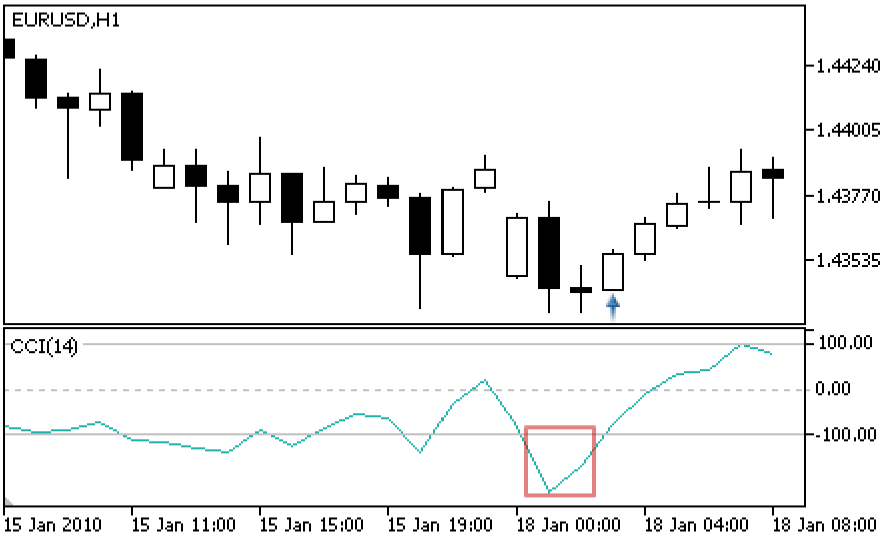
This module of signals is based on the market models of the oscillator Commodity Channel Index. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

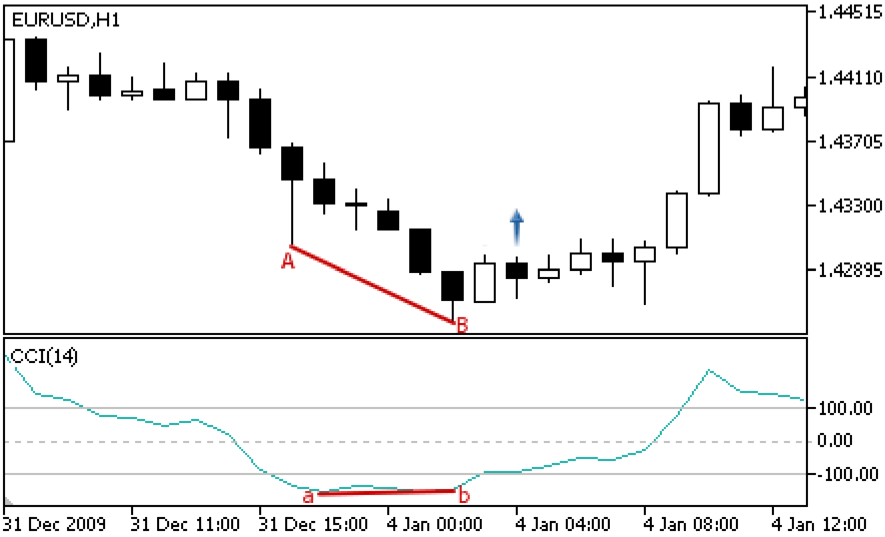
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the level of oversold level (default value is -100).



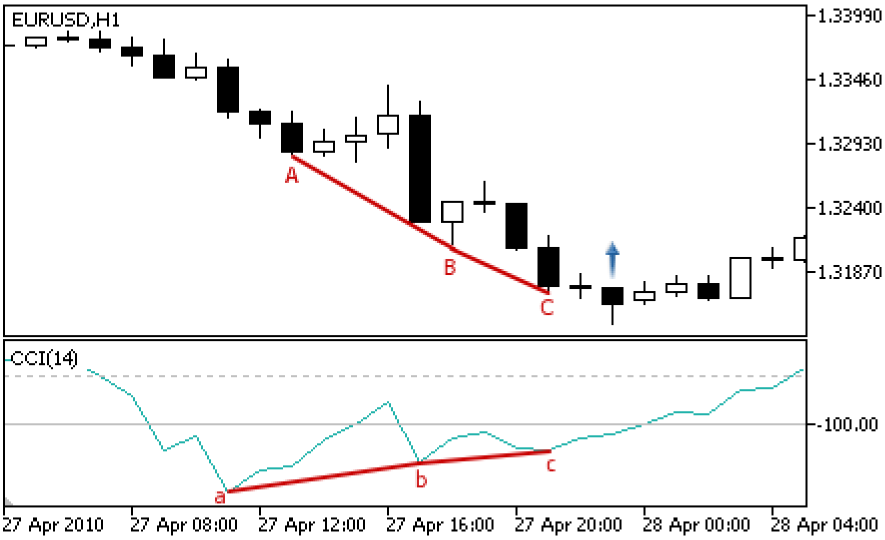
- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.



</td></tr>
</table>

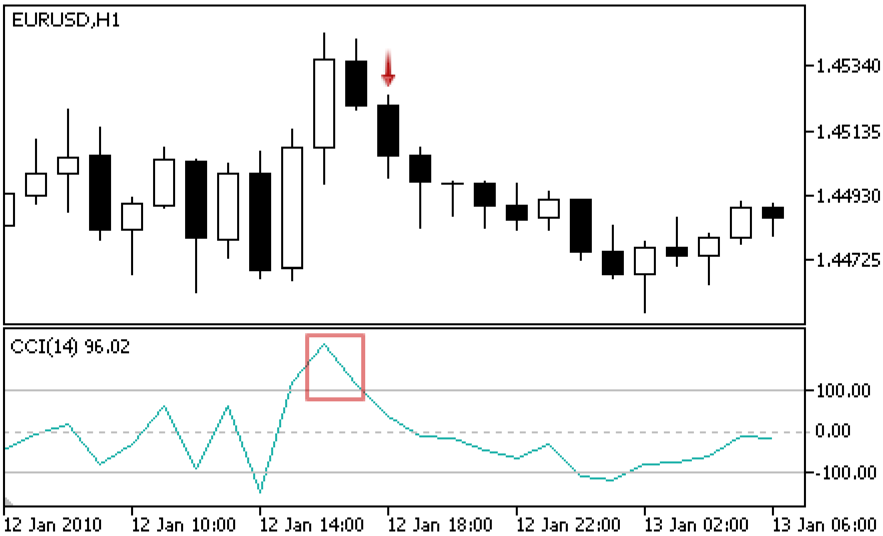
<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>

- **Double divergence** — the oscillator formed three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.

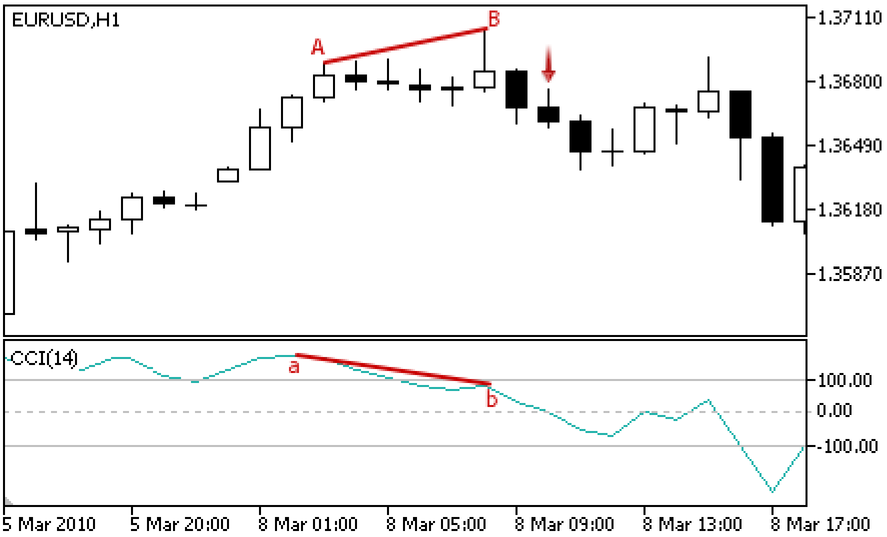


</td></tr>
<tr><td>For selling</td><td>

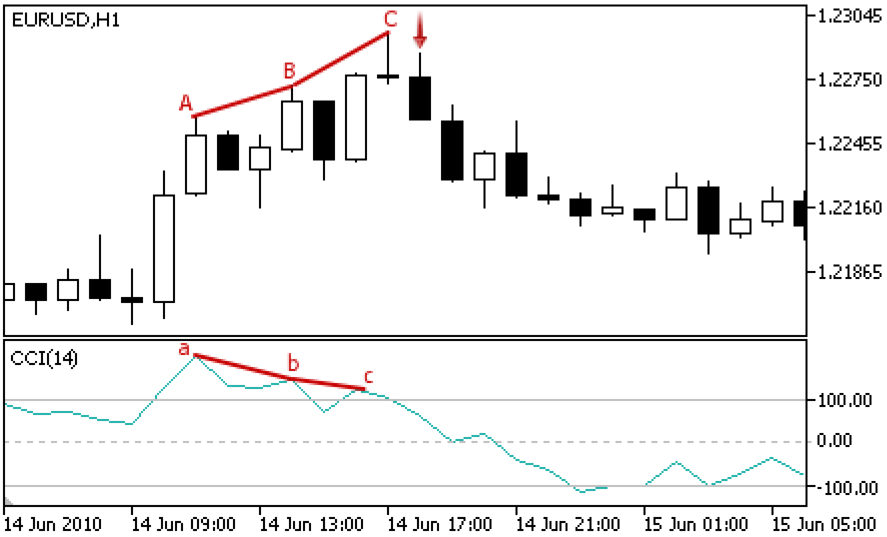
- **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 100).



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.
</td></tr>
</table>

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>


- **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.



</td></tr>
<tr><td>No objections to buying</td><td>Value of the oscillator grows at the analyzed bar.</td></tr>
<tr><td>No objections to selling</td><td>Value of the oscillator falls at the analyzed bar.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodCCI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

Signals of the Oscillator DeMarker

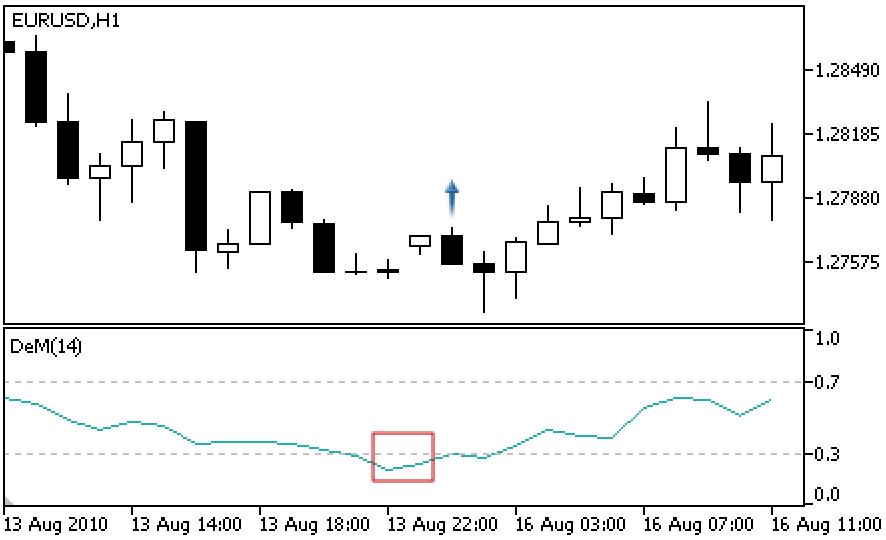
This module of signals is based on the market models of the oscillator DeMarker. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

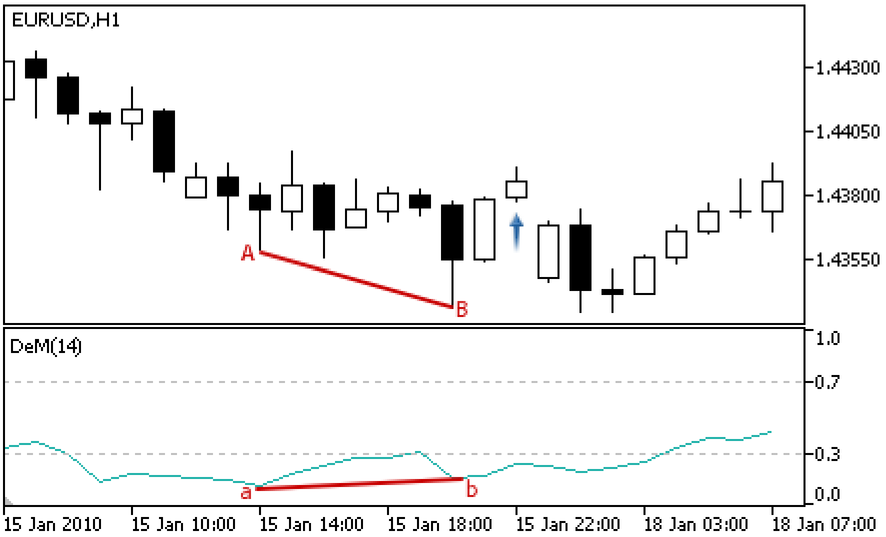
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is 0.3).

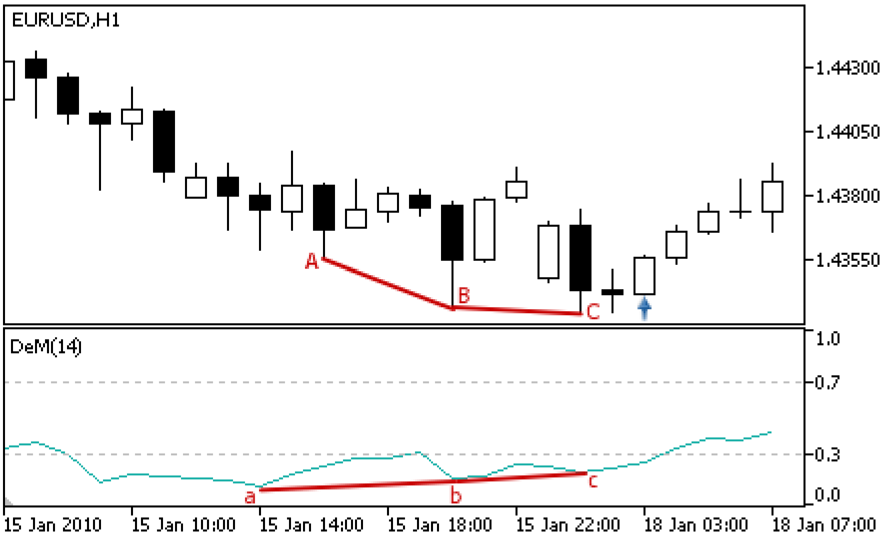
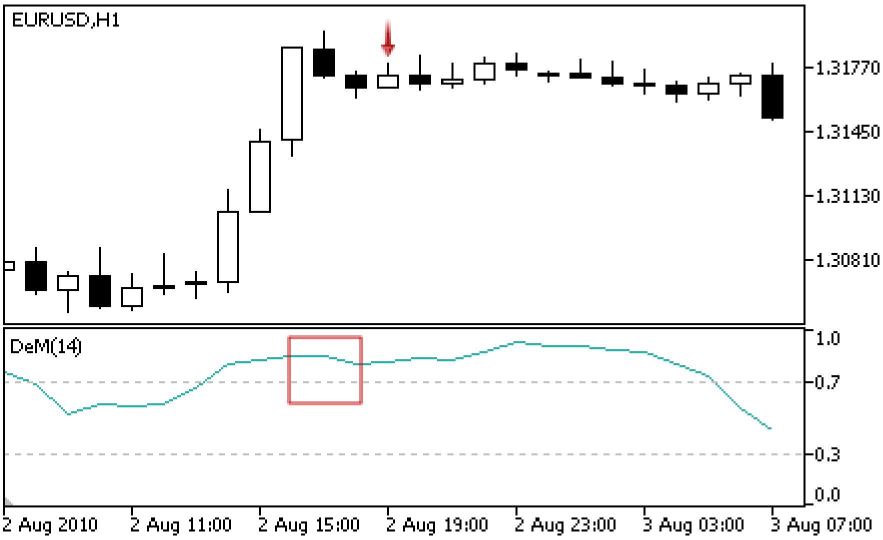


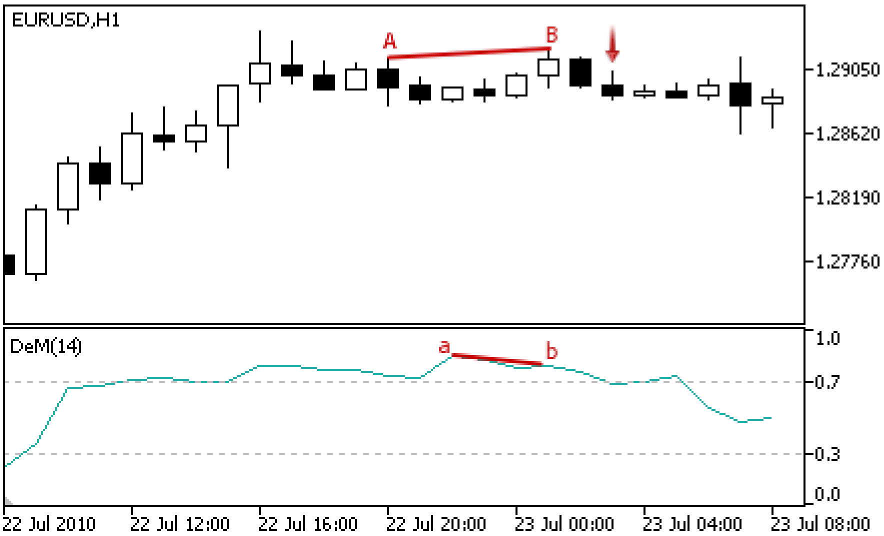
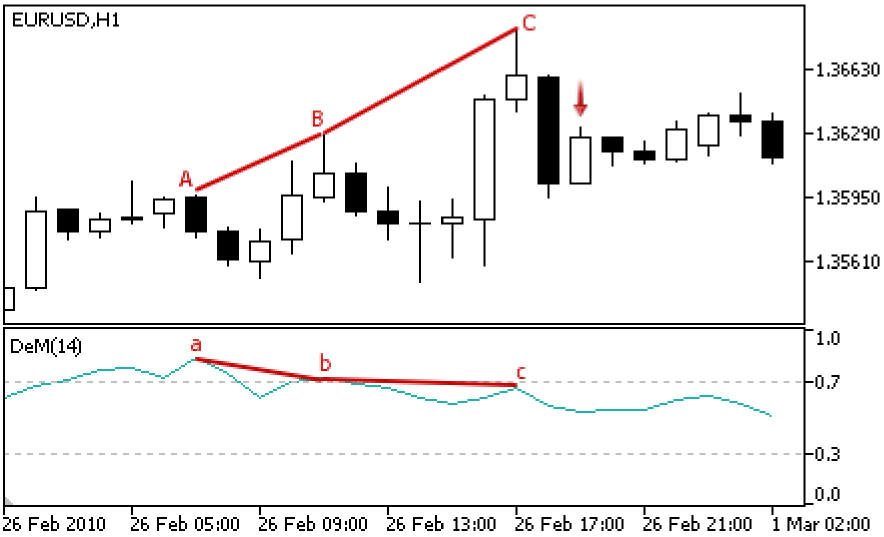
- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.



- **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed

</td></tr>
</table>

Signal Type	Description of Conditions
	three corresponding bottoms, and each of them is lower than the previous one. 
For selling	- **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 0.7).  - **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodDeM	Period of calculation of the oscillator.

Signals of the Indicator Double Exponential Moving Average

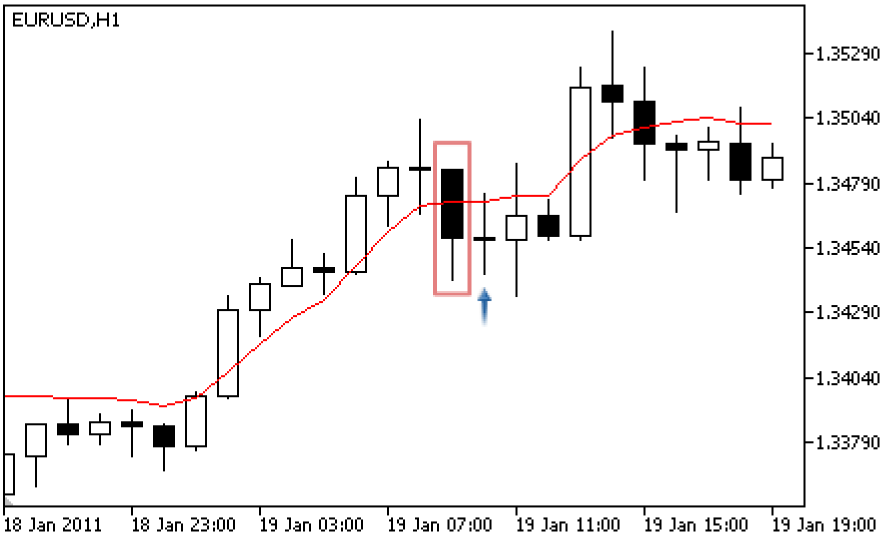
This module is based on the market models of the indicator Double Exponential Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

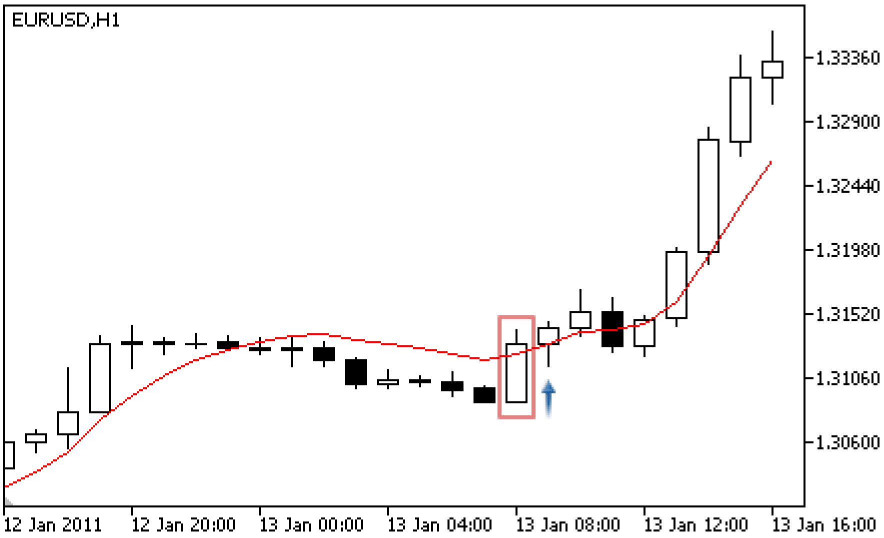
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Failed breakout.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).



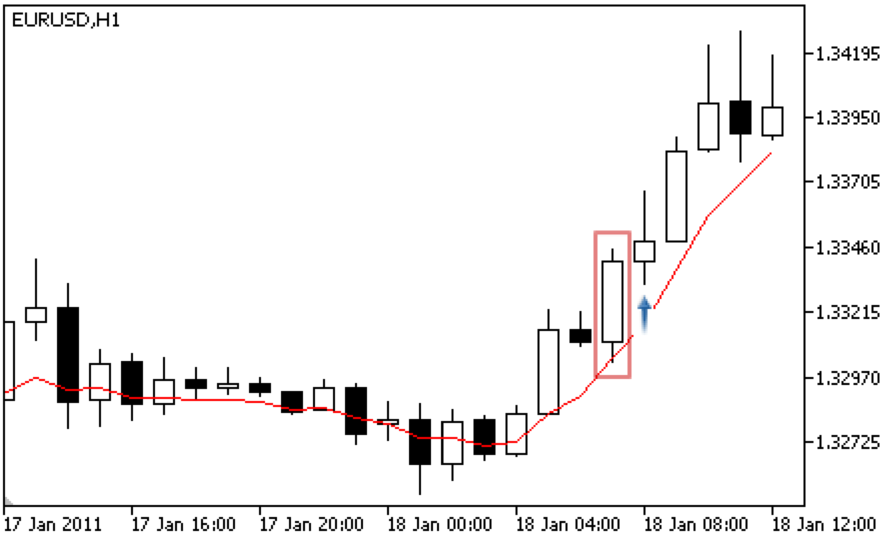
- **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal).



</td></tr>
</table>

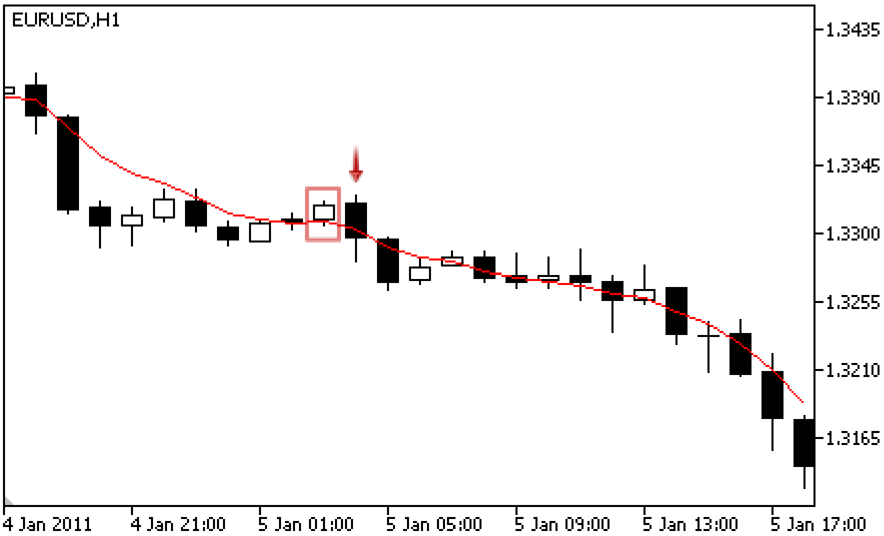
<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>

- **Formed breakout.** The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (signal for a roll-back from the indicator line).



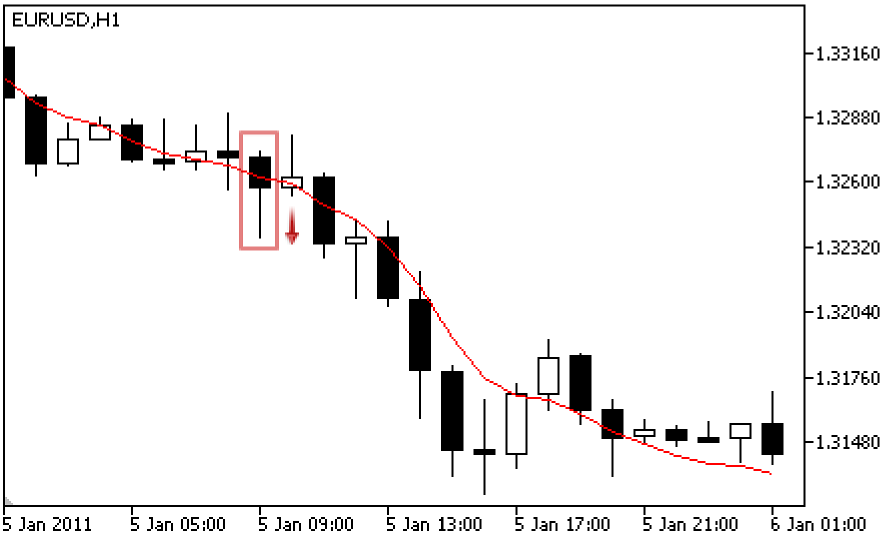
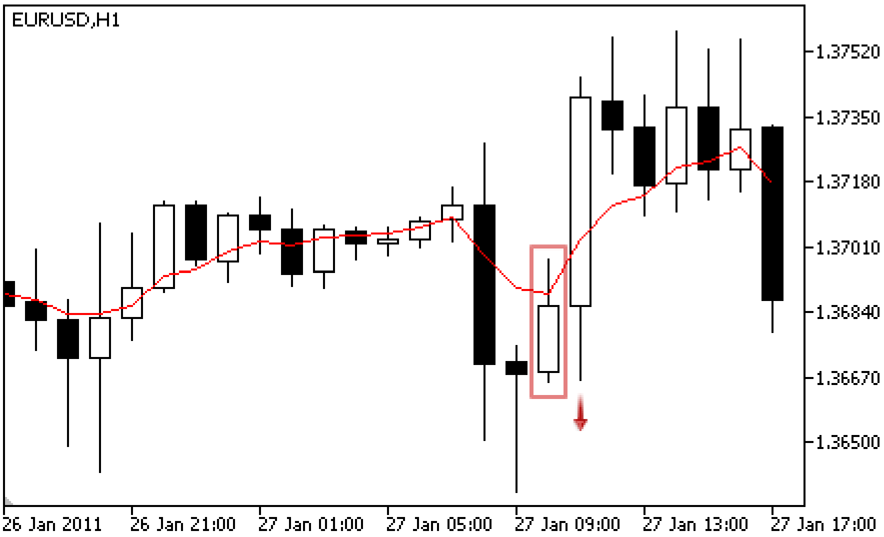
</td></tr>
<tr><td>For selling</td><td>

- **Failed breakout.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).



- **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

</td></tr>
</table>

Signal Type	Description of Conditions
	 - **Formed breakout.** The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

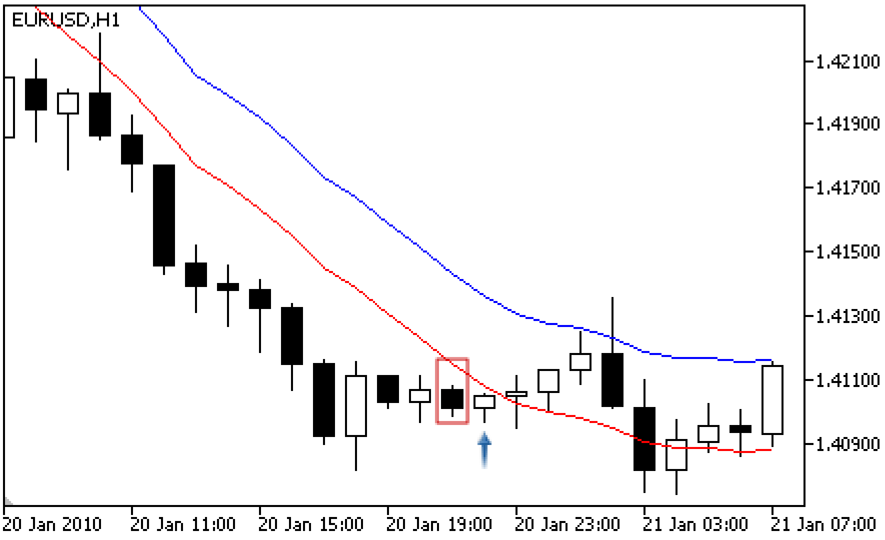
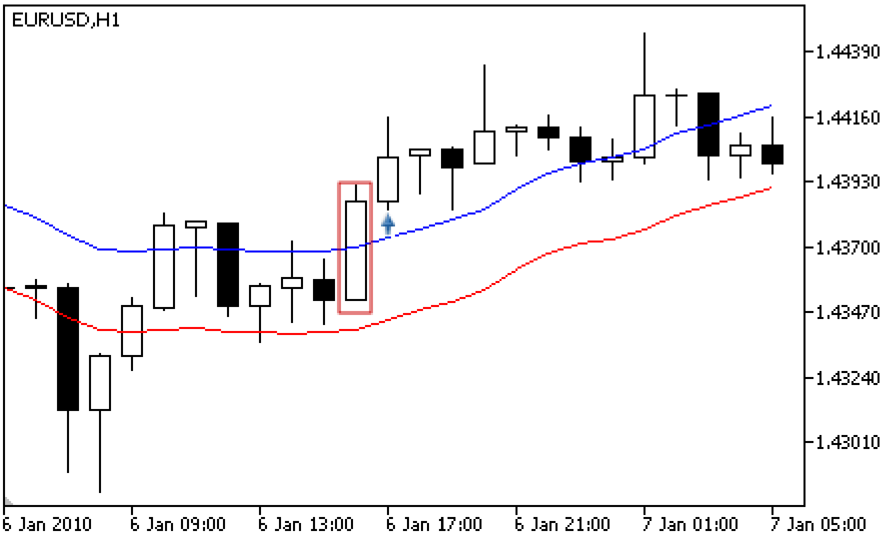
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

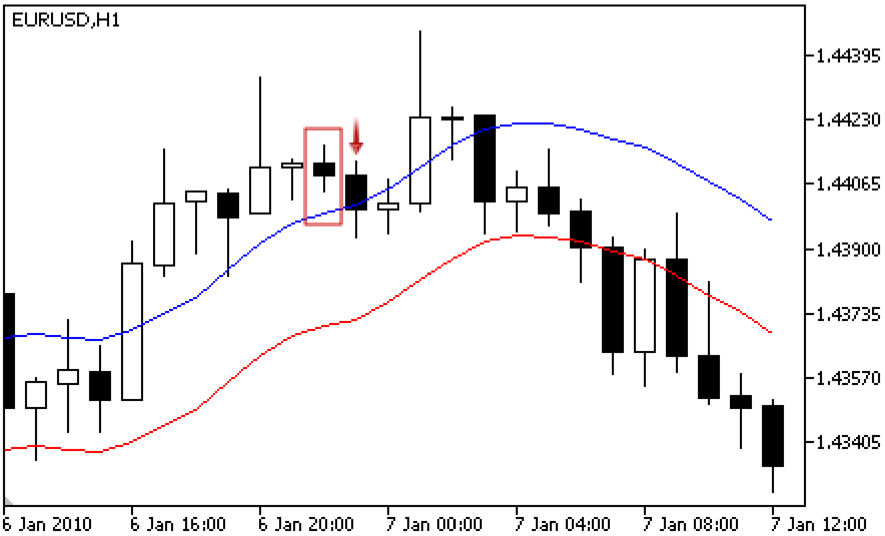
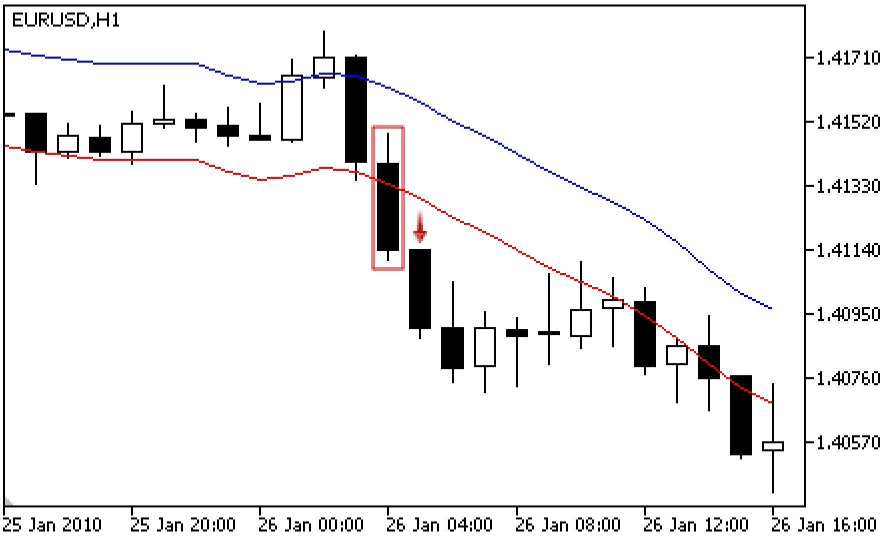
Signals of the Indicator Envelopes

This module of signals is based on the market models of the indicator Envelopes. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- The price is near the lower line of the indicator at the analyzed bar.  - The price crossed the upper line of the indicator at the analyzed bar. 
For selling	- The price is near the upper line of the indicator at the analyzed bar.

Signal Type	Description of Conditions
	 - The price crossed the lower line of the indicator at the analyzed bar. 
No objections to buying	No signals.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

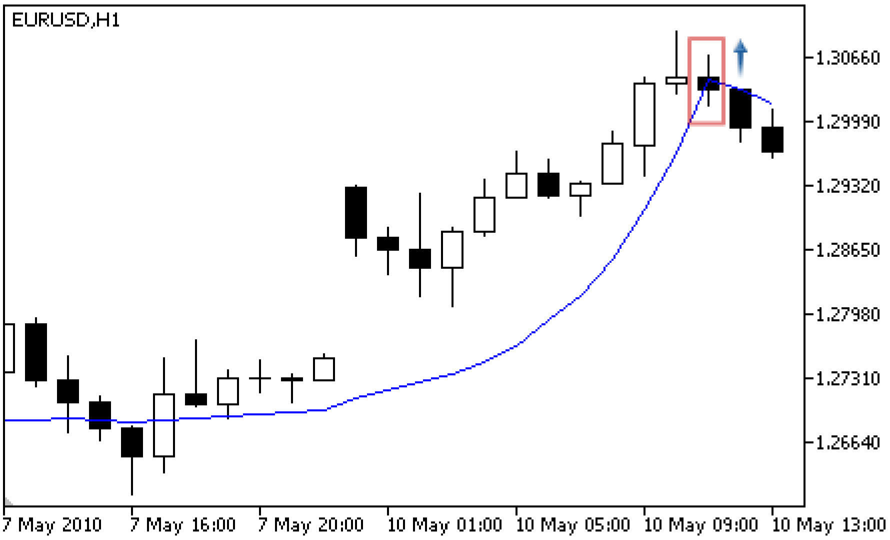
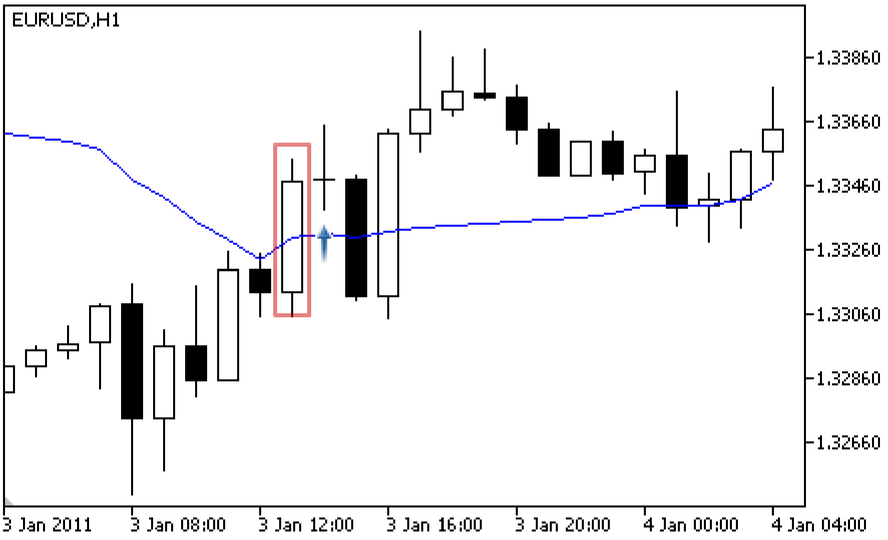
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of calculation of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.
Deviation	Deviation of the envelope borders from the center line (MA) in percentage terms.

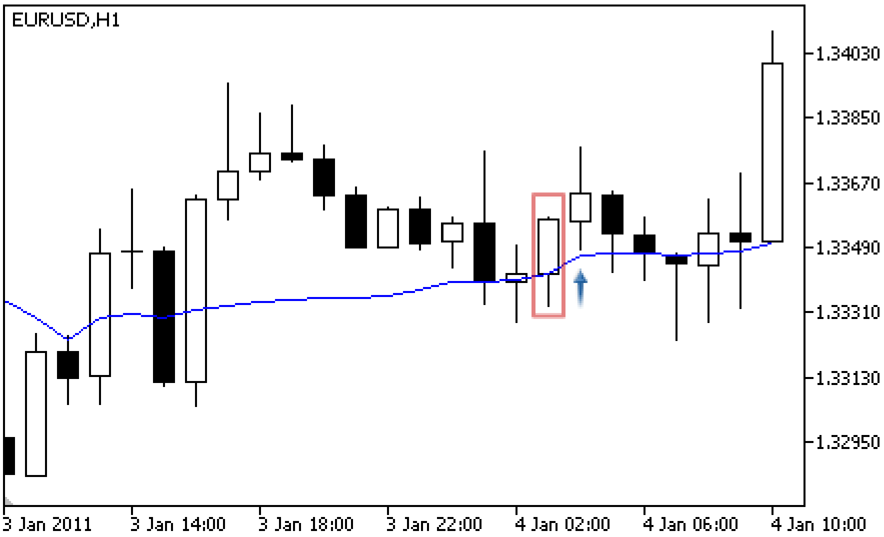
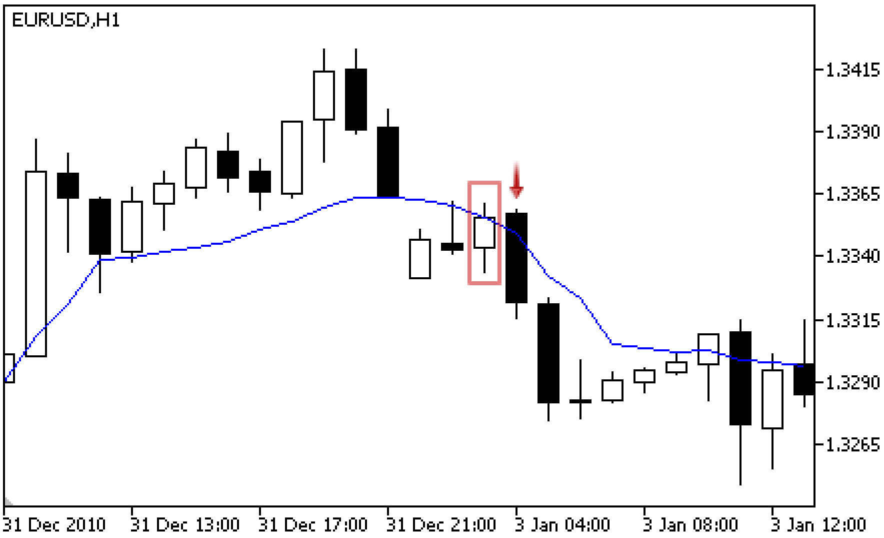
Signals of the Indicator Fractal Adaptive Moving Average

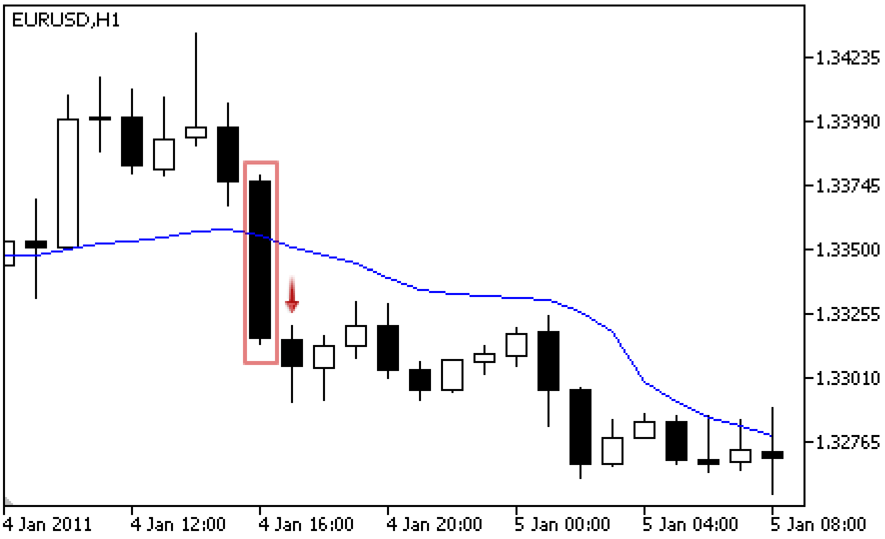
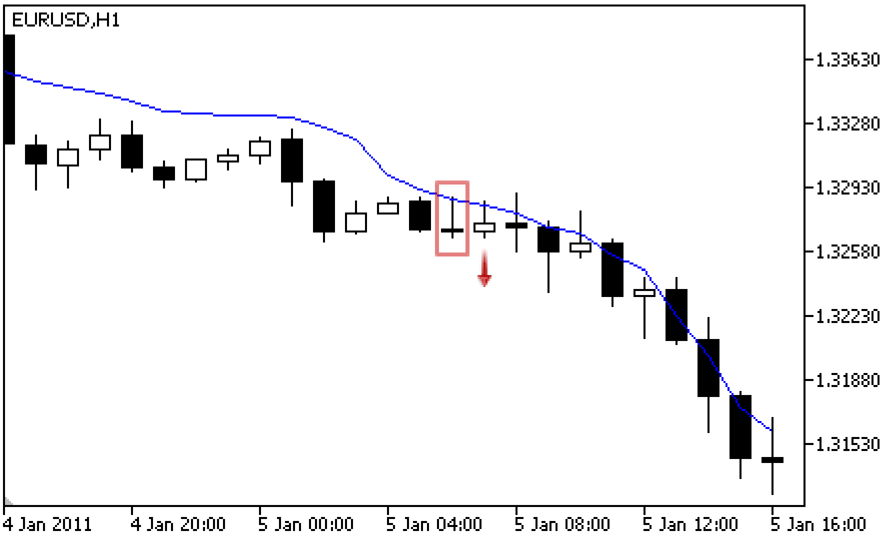
This module of signals is based on the market models of the indicator Fractal Adaptive Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- **Failed breakout.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).  - **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	- **Formed breakout.** The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (signal for a roll-back from the indicator line). 
For selling	- **Failed breakout.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line).  - **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 • **Formed breakout.** The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Intraday Time Filter

This module is based on the assumption that the efficiency of market models changes in time. Using this module, you can filter signals received from the other modules by hour and days of week. It allows increasing the quality of generated signals due to cutting off the unfavorable time periods. The mechanism of making trade decisions on the basis of signals of the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	No signals.
For selling	No signals.
No objections to buying	The current date and time meet the specified parameters.
No objections to selling	The current date and time meet the specified parameters.

Adjustable Parameters

This module has the following adjustable parameters:

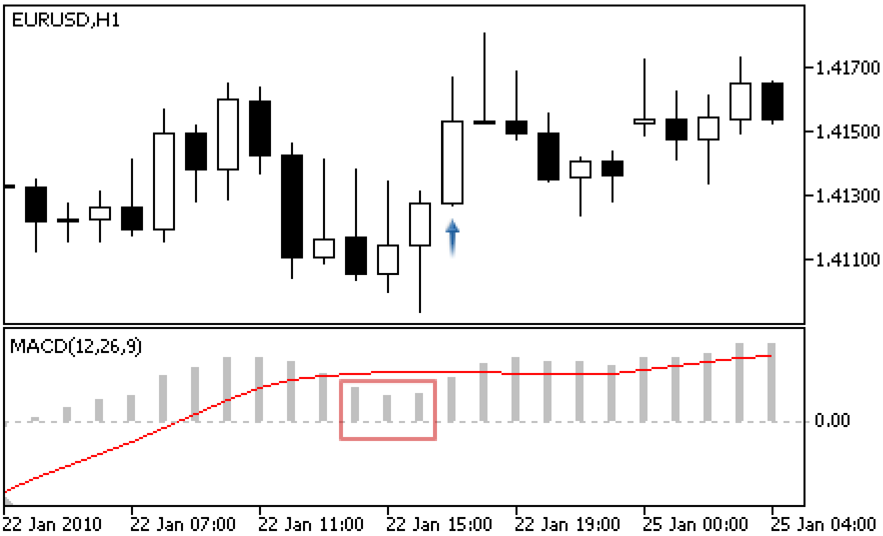
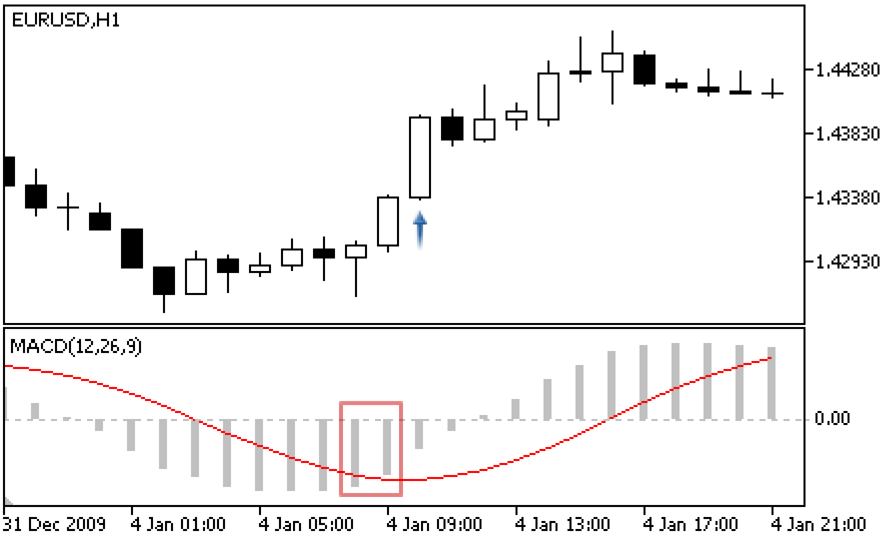
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
GoodHourOfDay	Number of the only hour of day (from 0 to 23) when trade signals will be enabled. If the value is -1, the signals will be enabled through the whole day.
BadHoursOfDay	The bit field. Each bit of this field corresponds to an hour of day (0 bit - 0 hour, ..., 23 bit - 23-rd hour). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding hour. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding hour. A specified number is represented as a binary number and is used as bit mask. Disabled hours have higher priority than the enabled ones.
GoodDayOfWeek	Number of the only day of week (from 0 to 6, where 0 is Sunday), when trade signals will be enabled. If the value is -1, the signals will be enabled on any day.
BadDaysOfWeek	The bit field. Each bit of this field corresponds to a day of week (0 bit - Sunday, ..., 6 bit - Saturday). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding day. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding day. A specified number is represented as a binary number and is used as bit mask. Disabled days have higher priority than the enabled ones.

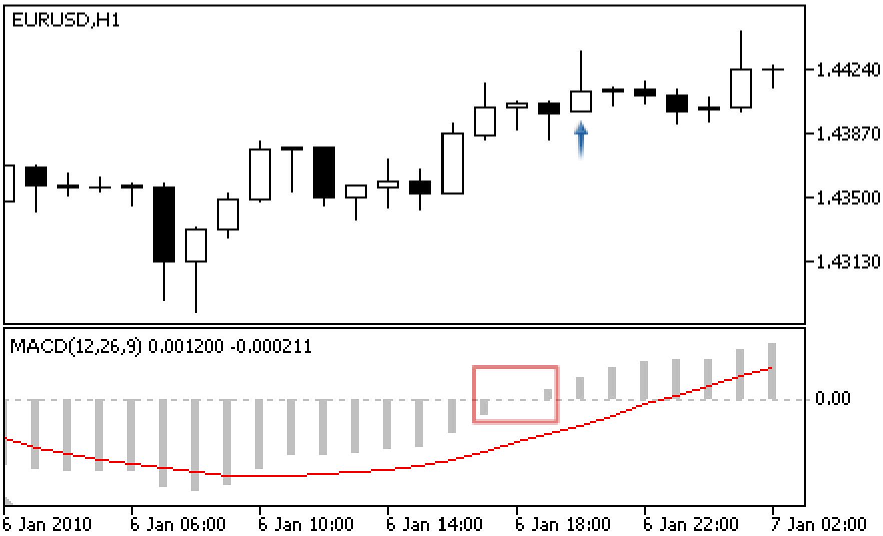
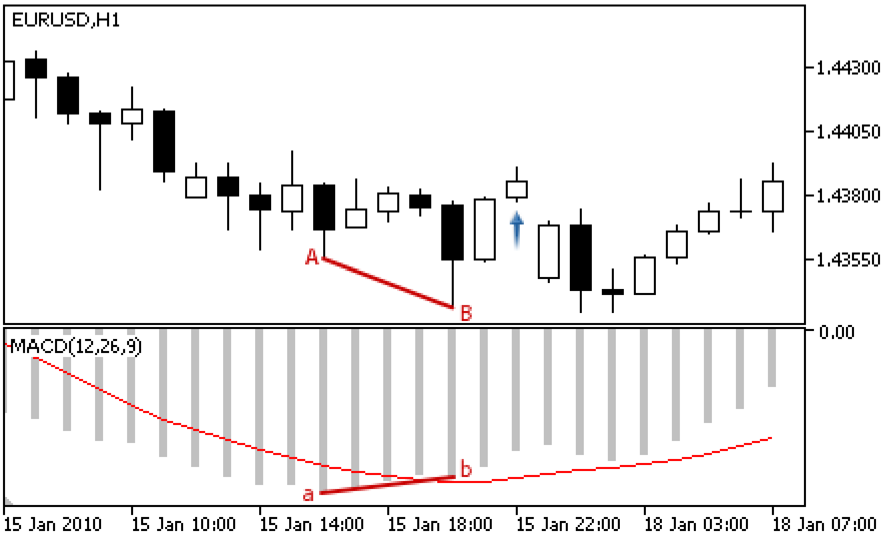
Signals of the Oscillator MACD

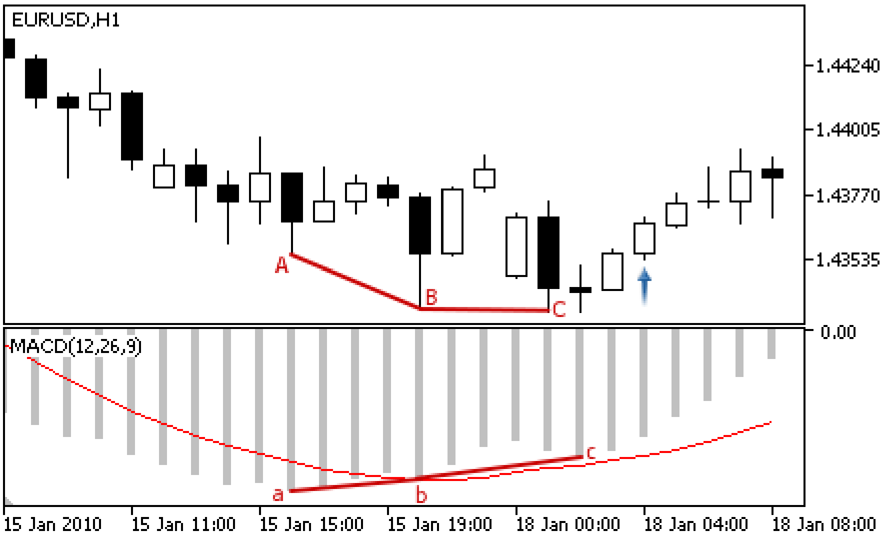
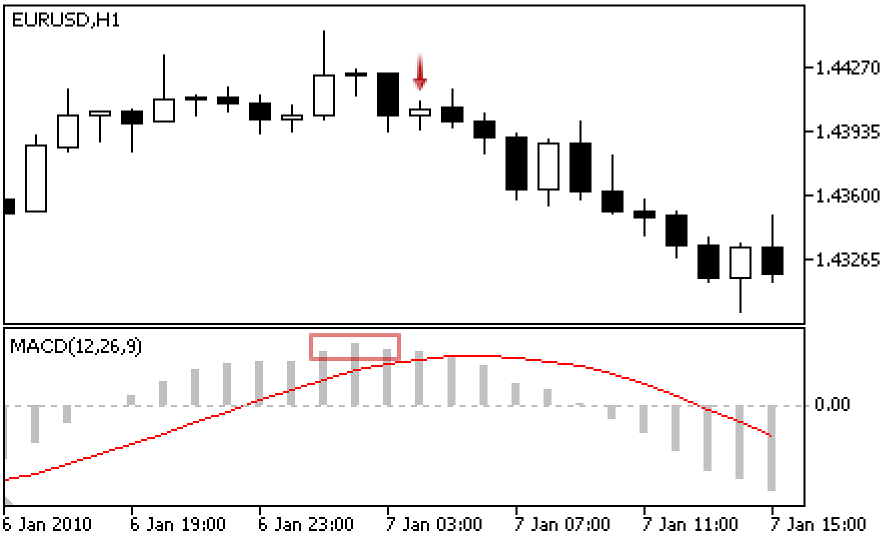
This module of signals is based on the market models of the oscillator MACD. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

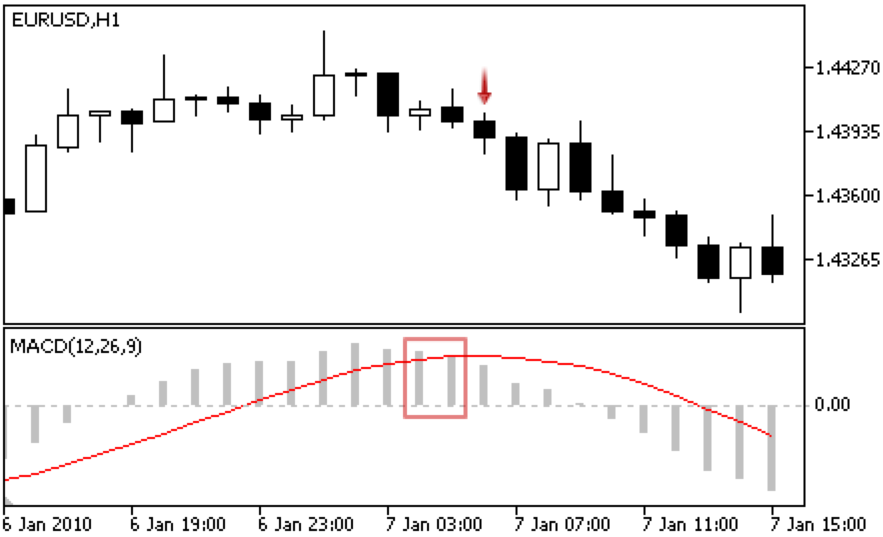
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  - **Crossover of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one.  - **Crossing the zero level** — the main line is above the zero level at the analyzed bar and below the zero level at the previous one.

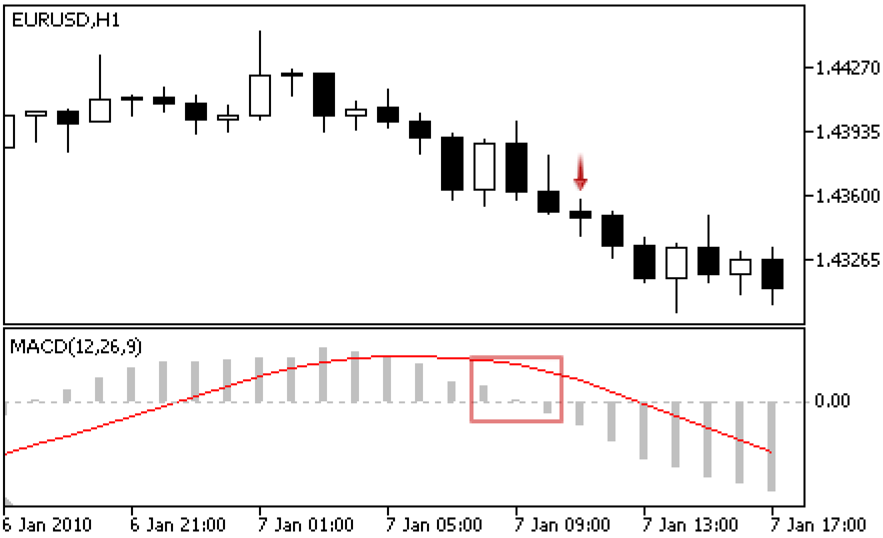
Signal Type	Description of Conditions
	 • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.  • **Double divergence** — the oscillator forms three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.

Signal Type	Description of Conditions
	
For selling	- **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  - **Crossover of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>

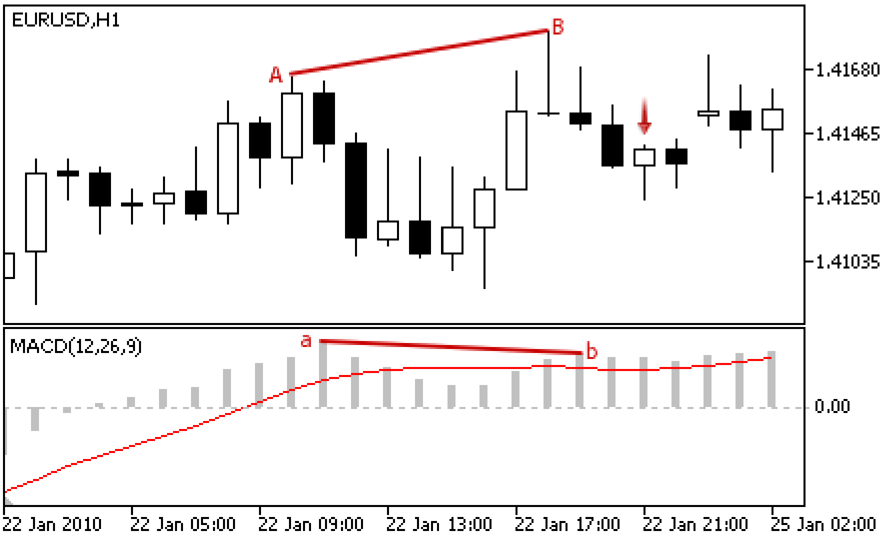
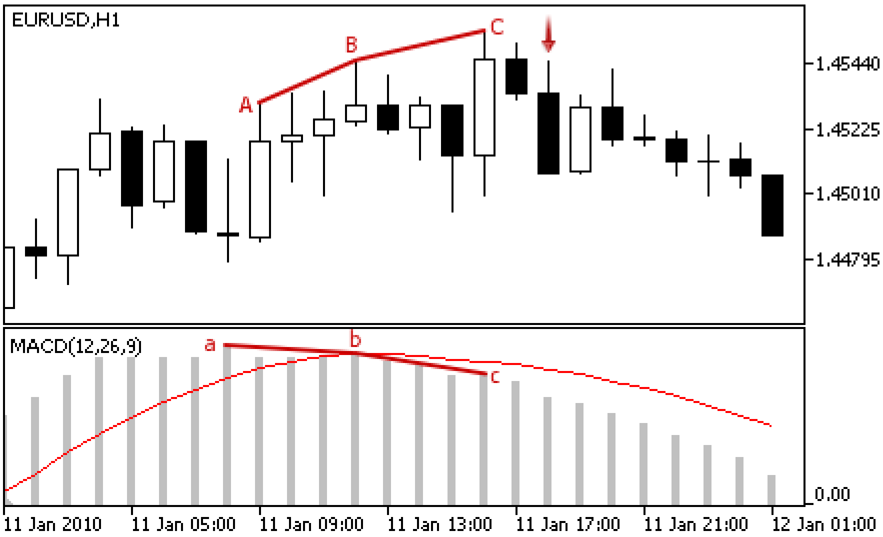


- **Crossing the zero level** — the main line is below the zero level at the analyzed bar and above the zero level at the previous one.



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

</td></tr>
</table>

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodFast	Period of calculation of the fast EMA.
PeriodSlow	Period of calculation of the slow EMA.
PeriodSignal	Period of smoothing.
Applied	A price series used for calculation of the oscillator.

Signals of the Indicator Moving Average

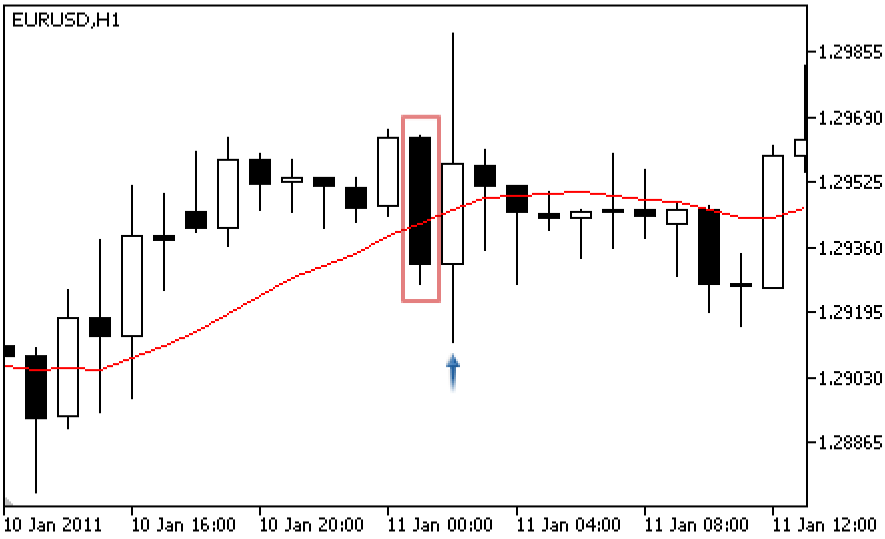
This module of signals is based on the market models of the indicator Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

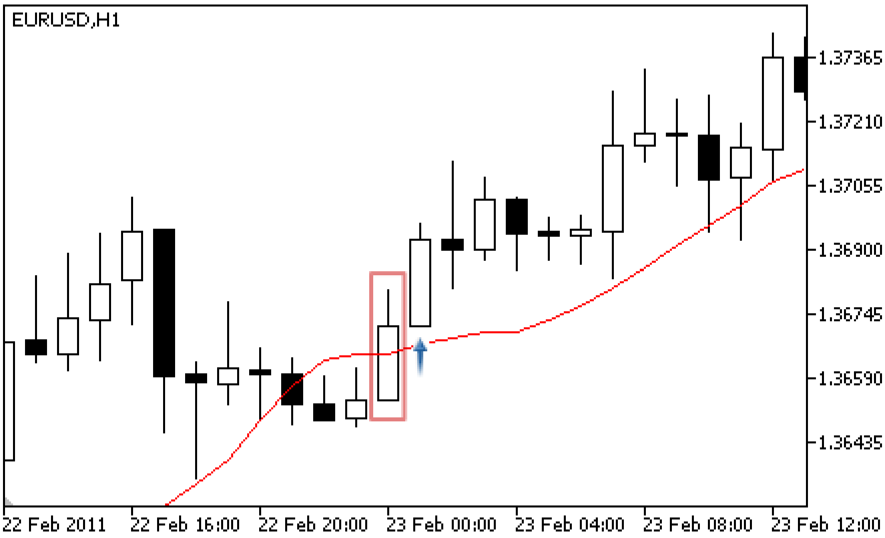
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

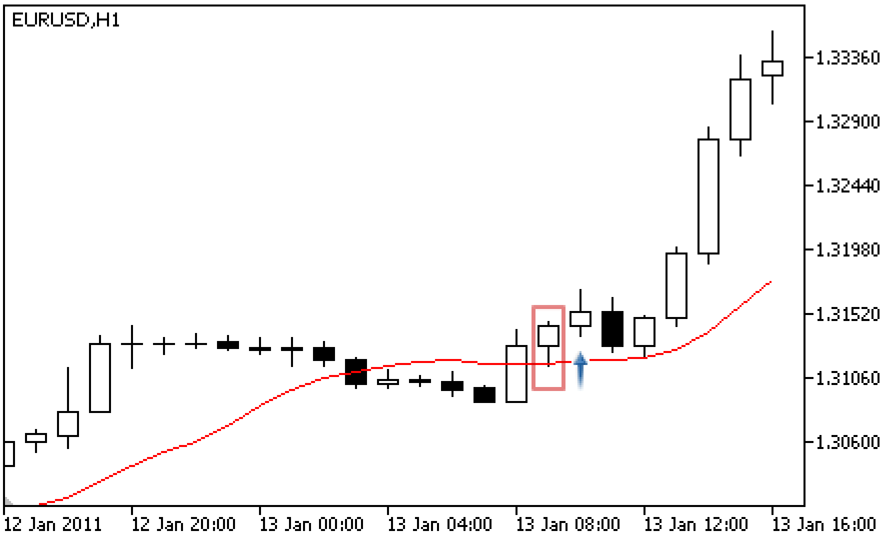
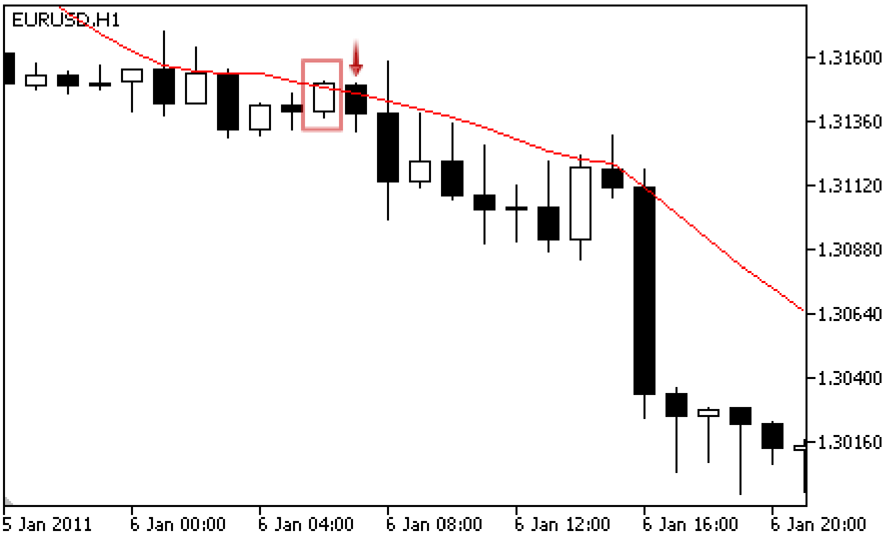
- **Failed breakout.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line).

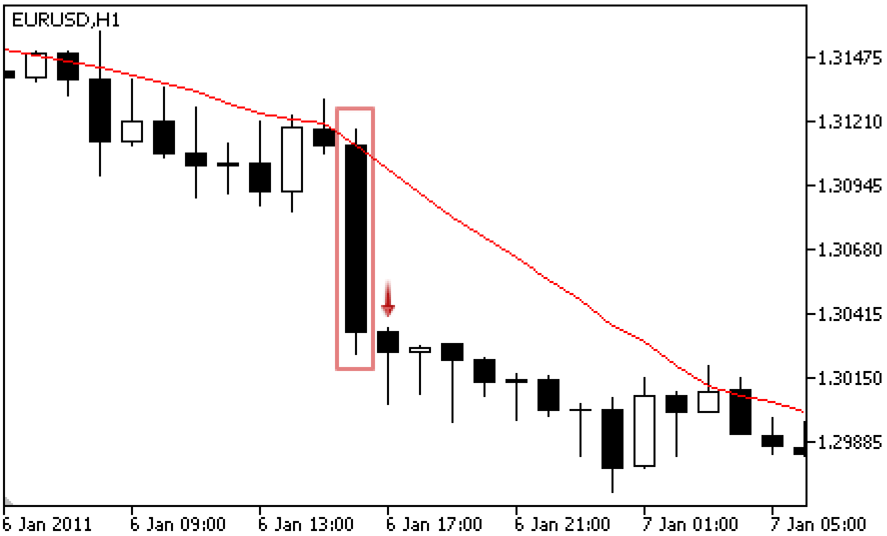
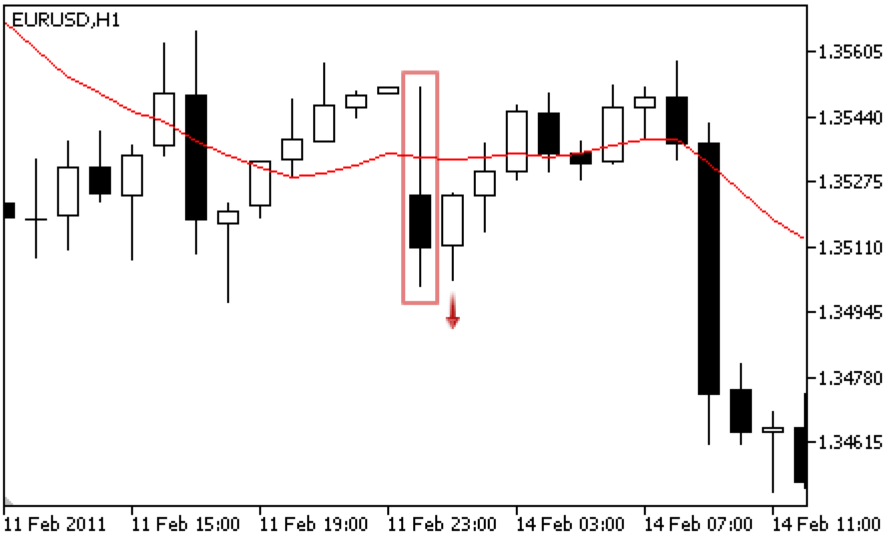


- **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal).



</td></tr>
</table>

Signal Type	Description of Conditions
	- **Formed breakout.** The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal for a roll-back from the indicator line). 
For selling	- **Failed breakout.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  - **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 - **Formed breakout.** The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal for a roll-back from the indicator line). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

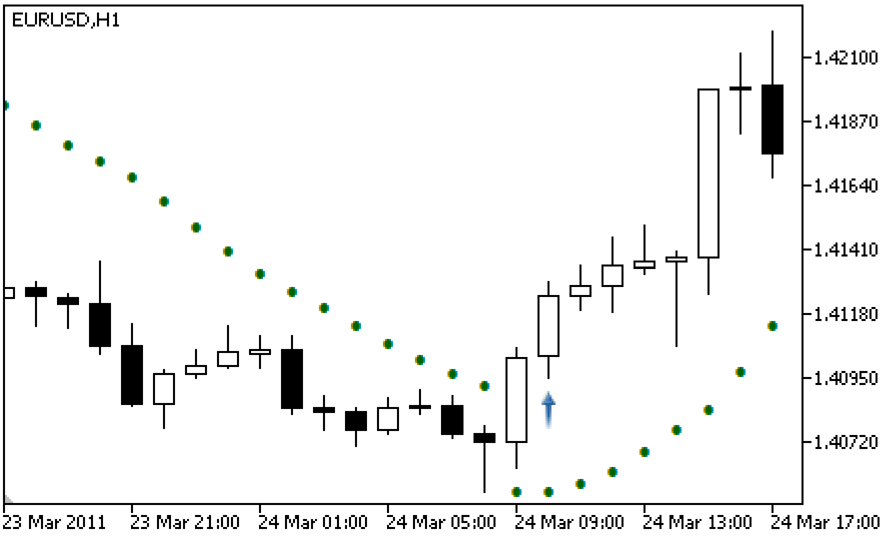
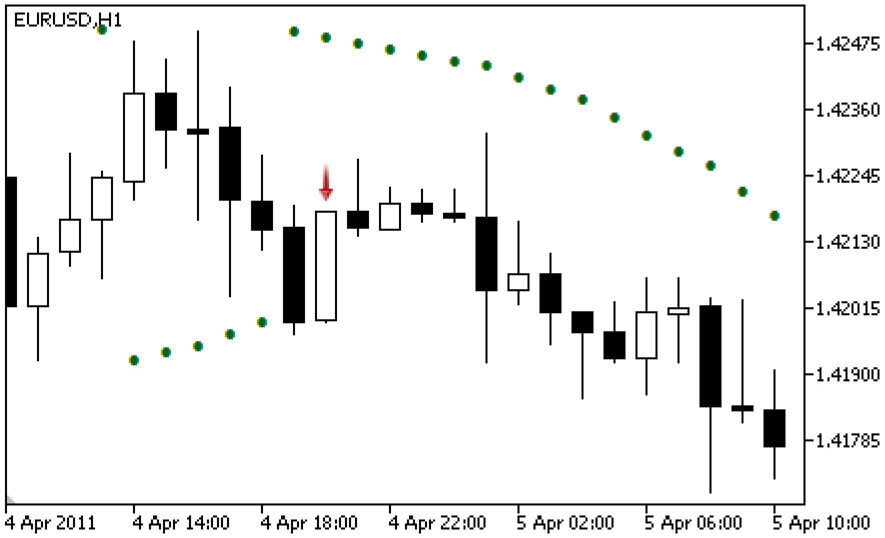
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the indicator Parabolic SAR

This module of signals is based on the market models of the indicator Parabolic SAR. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	**Reverse** — the indicator is below the price at the analyzed bar and above the price at the previous one. 
For selling	**Reverse** — the indicator is above the price at the analyzed bar and below the price at the previous one. 
No objections to buying	The price is above the indicator.

Signal Type	Description of Conditions
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

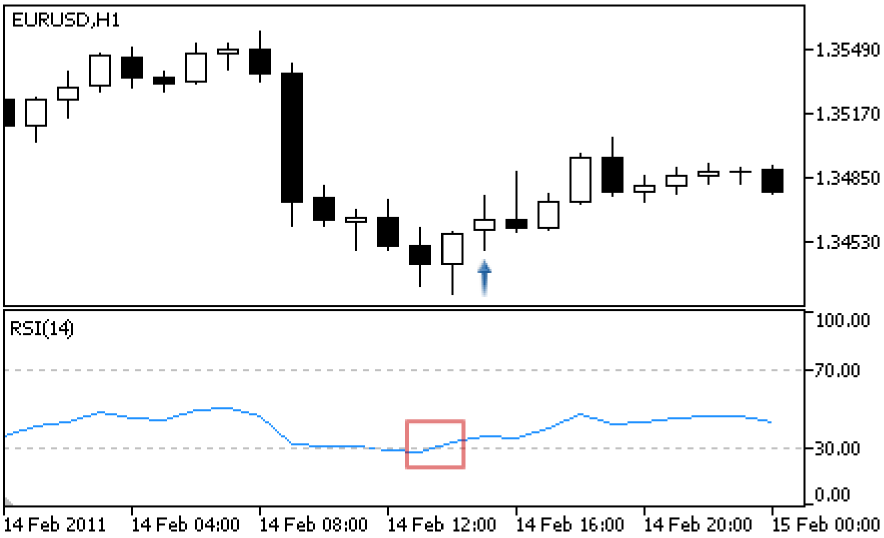
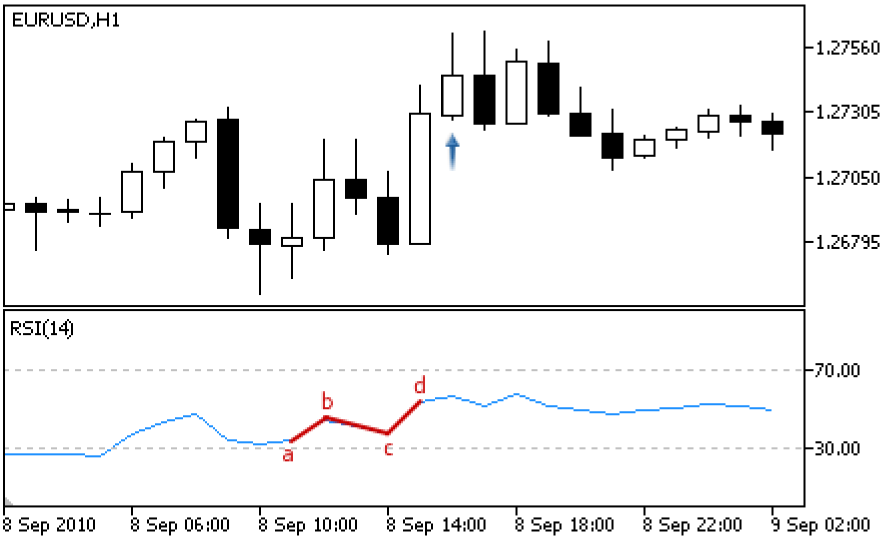
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
Step	The step of price increment.
Maximum	Maximum rate of the speed of convergence of the indicator with the price.

Signals of the Oscillator Relative Strength Index

This module of signals is based on the market models of the oscillator [Relative Strength Index](). The mechanism of making trade decisions based on signals obtained from the modules is described in a [separate section]().

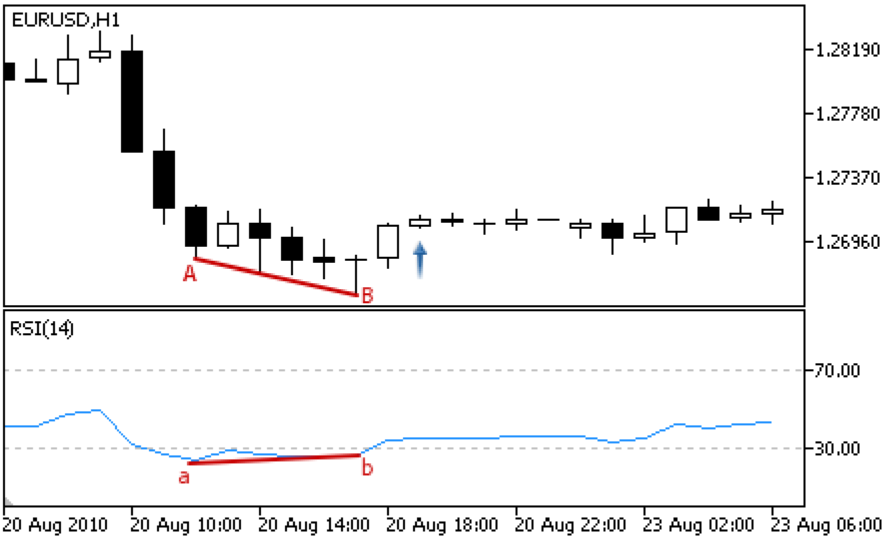
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

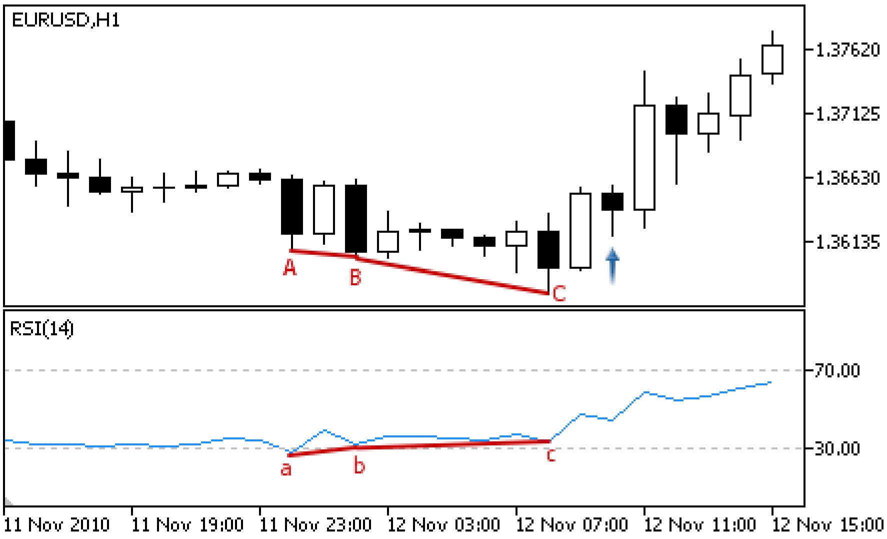
Signal Type	Description of Conditions
For buying	- **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is 30).  - **Failed swing** — the oscillator rises higher than the previous peak at the analyzed bar. 

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>

- **Divergence** — the first analyzed bottom of the oscillator is lower than the previous one, and the corresponding price bottom is lower than the previous one.

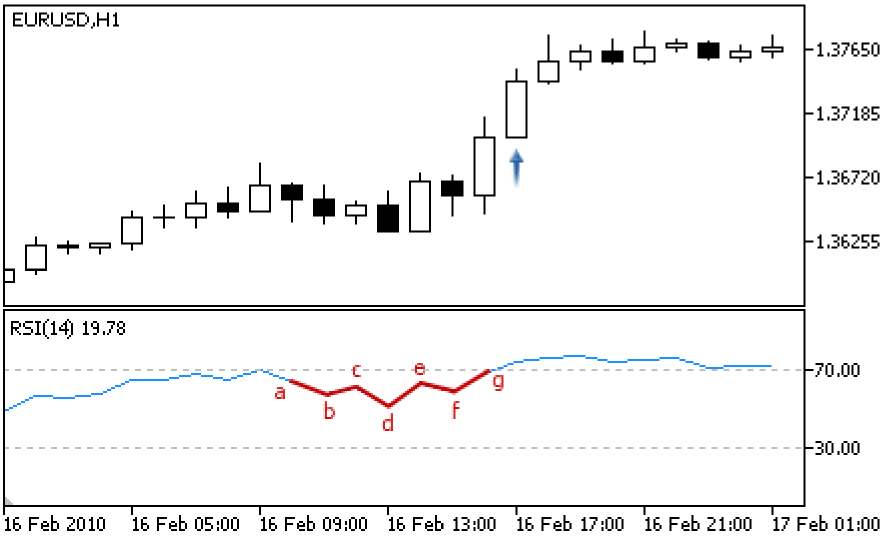
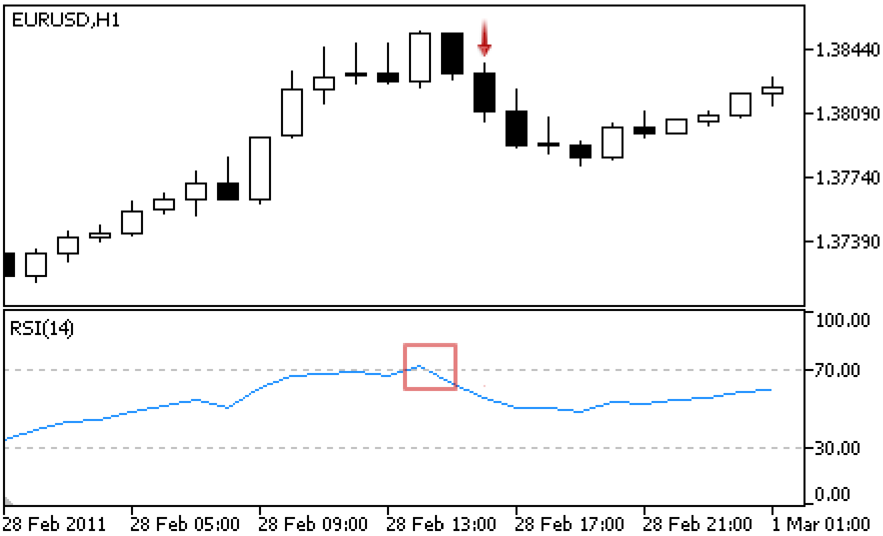


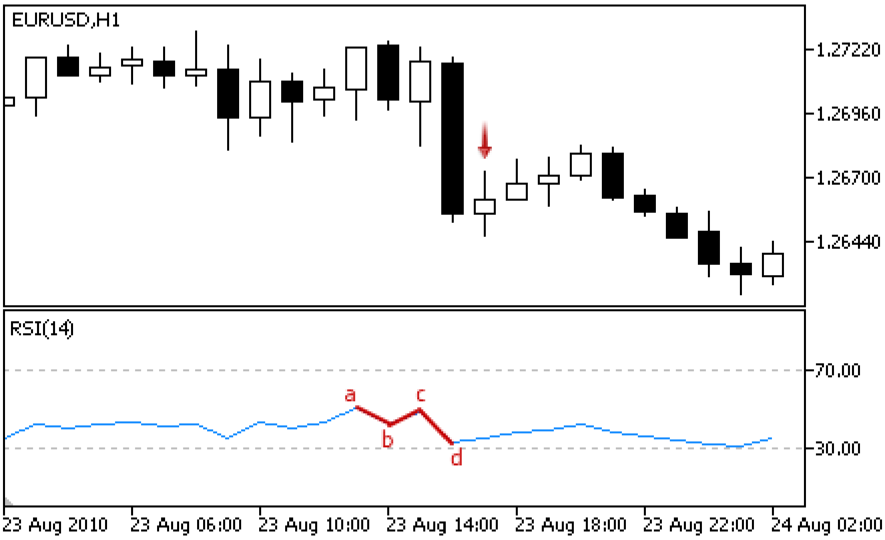
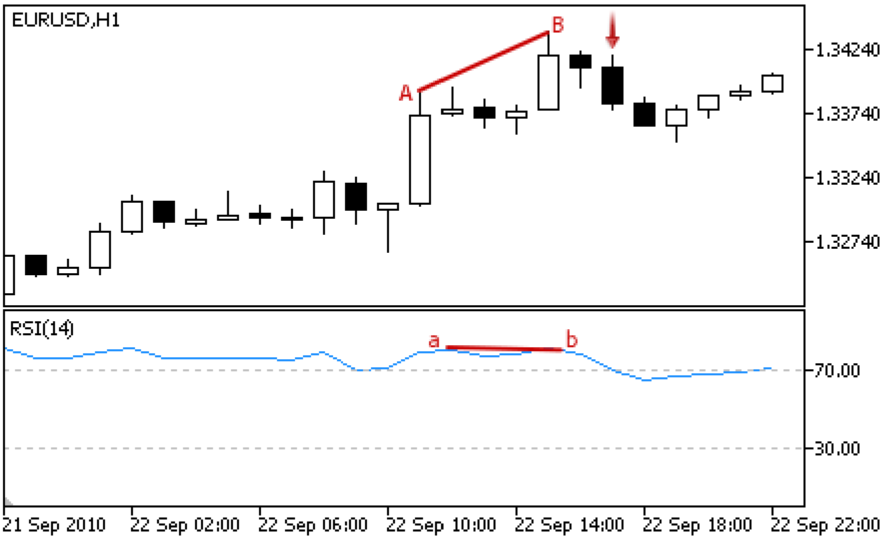
- **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.



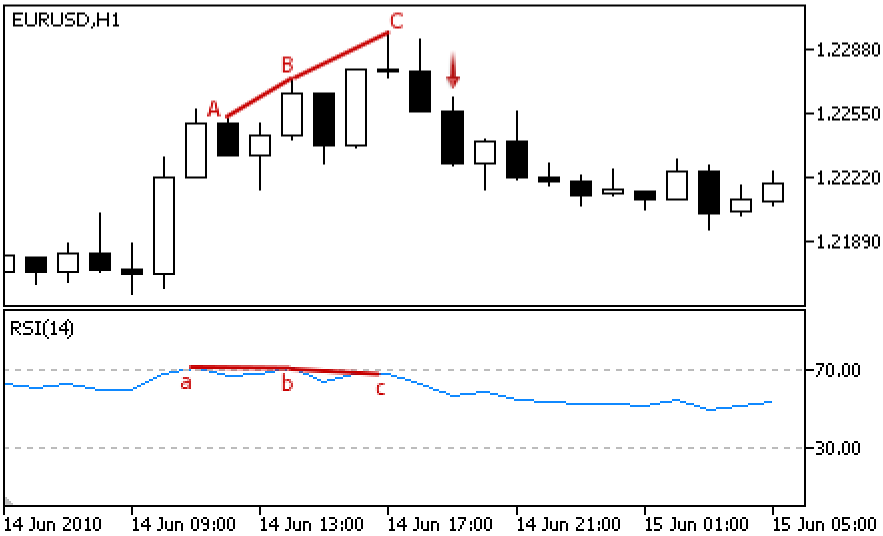
- **Head/Shoulders** — the oscillator formed three consequent bottoms, and the mid one is lower than the others.

</td></tr>
</table>

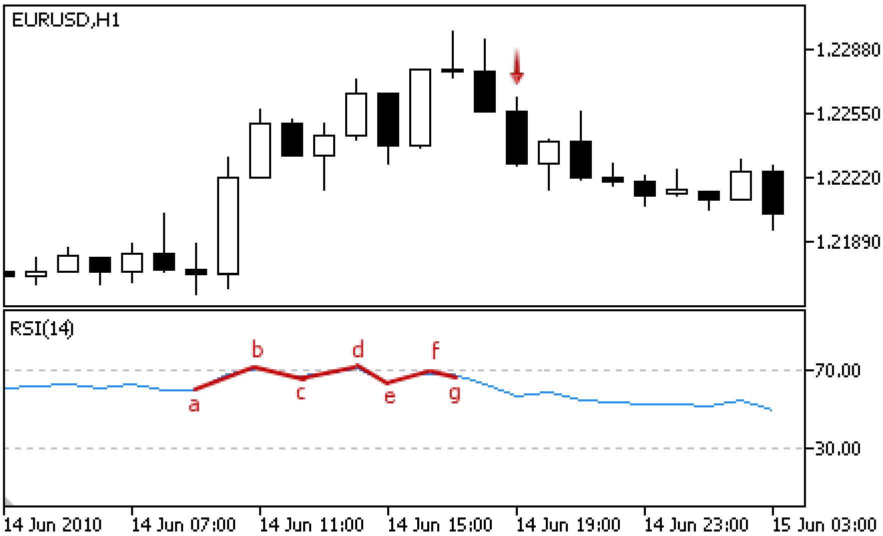
Signal Type	Description of Conditions
	
For selling	- **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 70).  - **Failed swing** — the oscillator falls lower than the previous bottom at the analyzed bar.

Signal Type	Description of Conditions
	 - **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.  - **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>



- **Head/Shoulders** — the oscillator formed three consequent peaks, and the mid one is higher than the others.



</td></tr>
<tr><td>No objections to buying</td><td>Value of the oscillator grows at the analyzed bar.</td></tr>
<tr><td>No objections to selling</td><td>Value of the oscillator falls at the analyzed bar.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

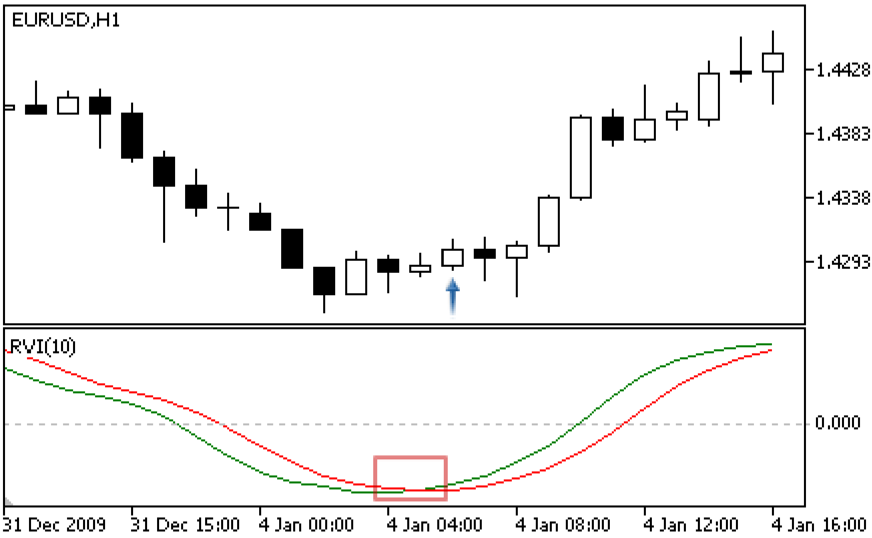
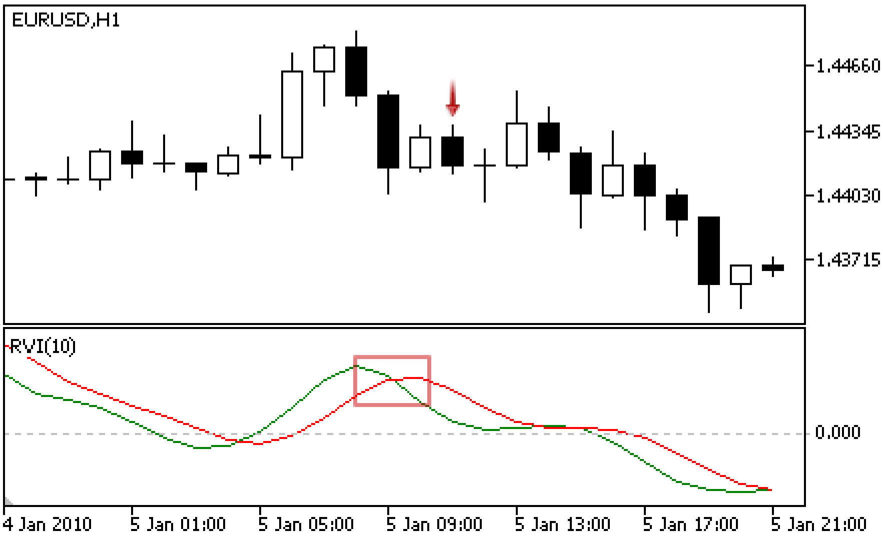
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRSI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

Signals of the Oscillator Relative Vigor Index

This module of signals is based on the market models of the oscillator Relative Vigor Index. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	**Crossing of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one. 
For selling	**Crossing of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.

Signal Type	Description of Conditions
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRVI	Period of calculation of the oscillator.

Signals of the Oscillator Stochastic

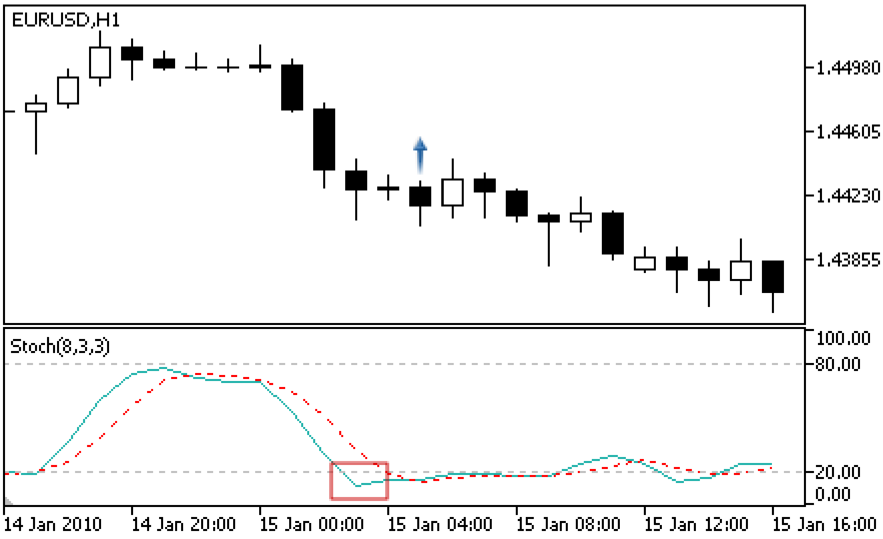
This module of signals is based on the market models of the oscillator Stochastic. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

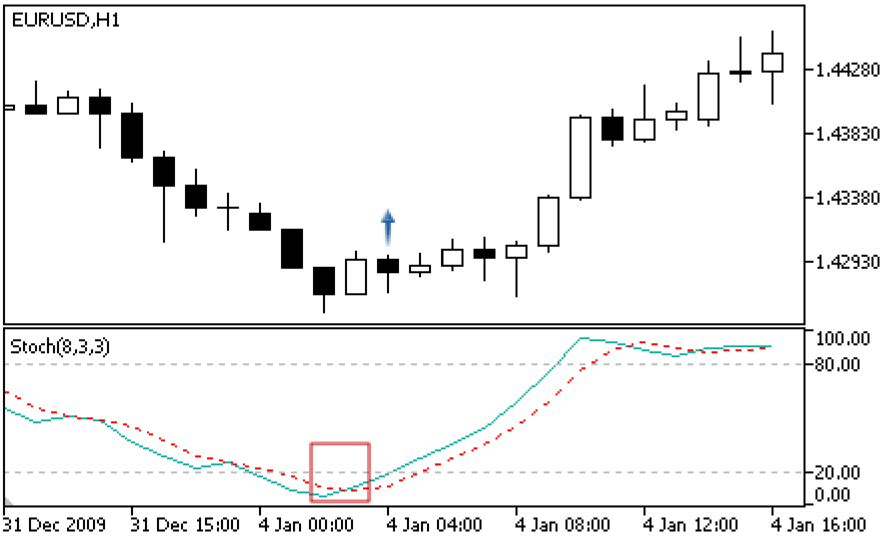
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

- **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).

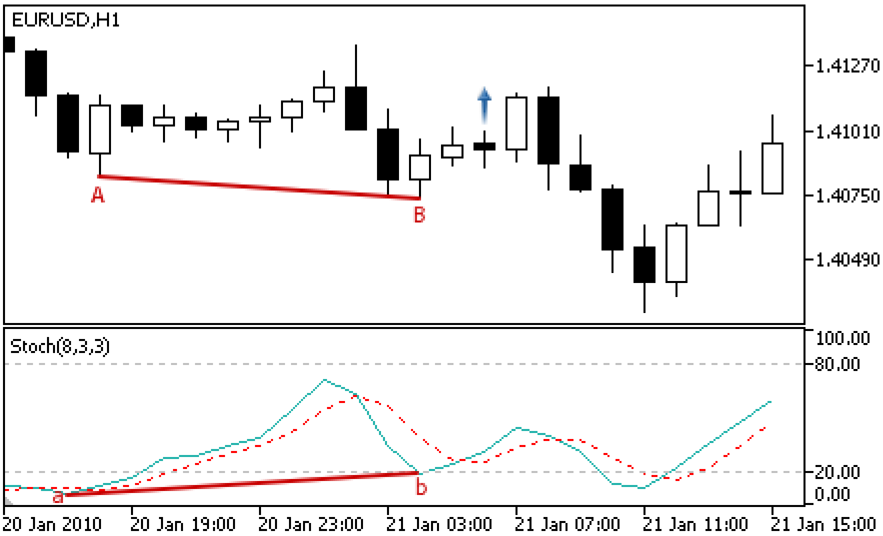


- **Crossing of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one.

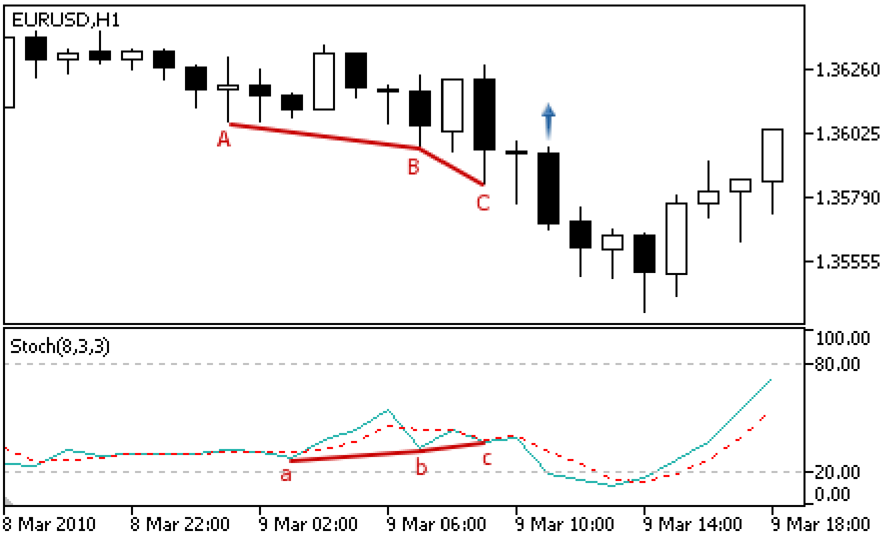


- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.

</td></tr>
</table>

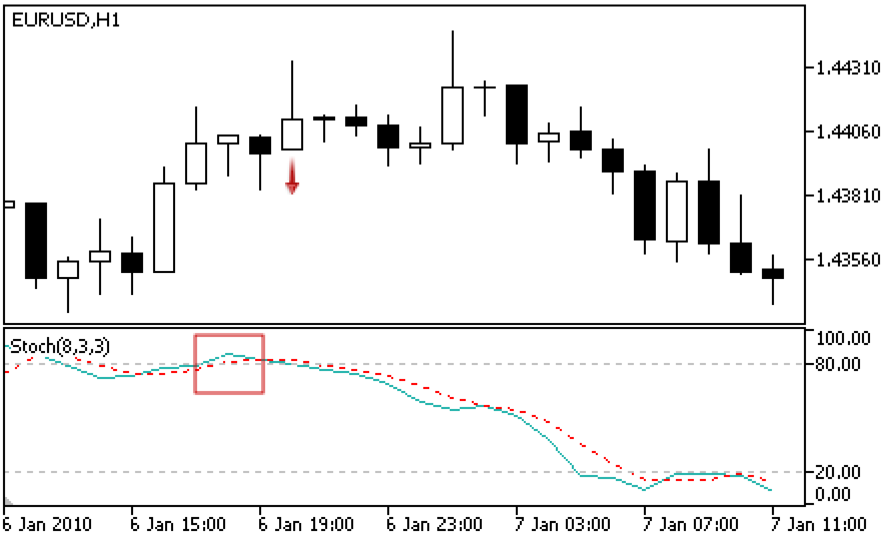
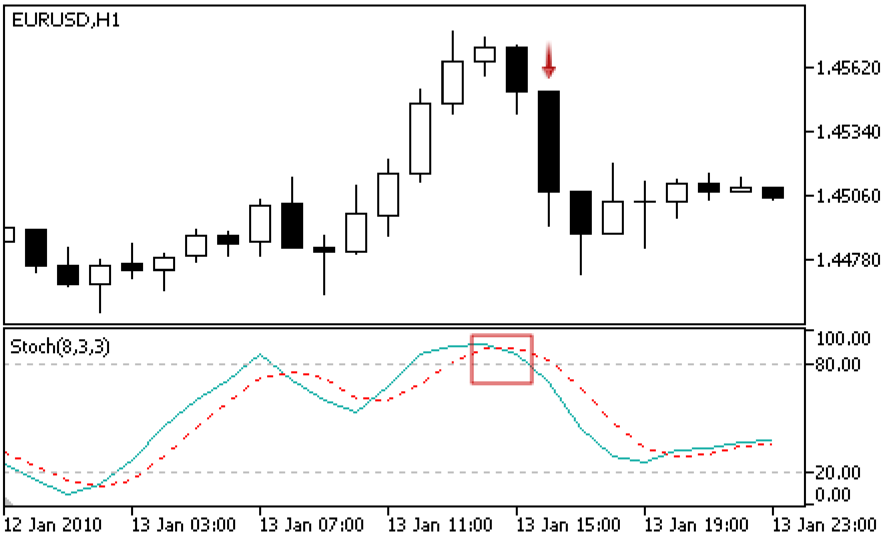
<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>


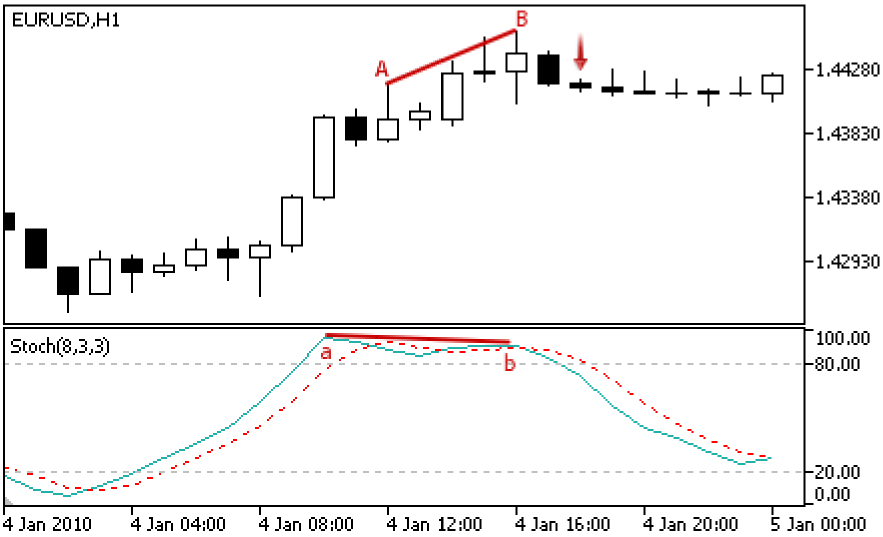
- **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.



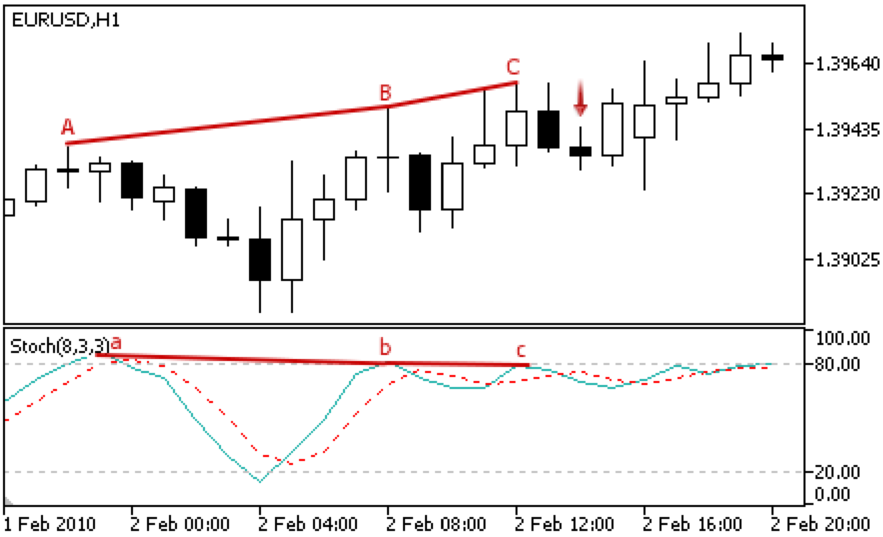
</td></tr>
<tr><td>For selling</td><td>

- **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).
</td></tr>
</table>

Signal Type	Description of Conditions
	 • **Crossing of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one.  • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>


- **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.



</td></tr>
<tr><td>No objections to buying</td><td>Value of the oscillator grows at the analyzed bar.</td></tr>
<tr><td>No objections to selling</td><td>Value of the oscillator falls at the analyzed bar.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

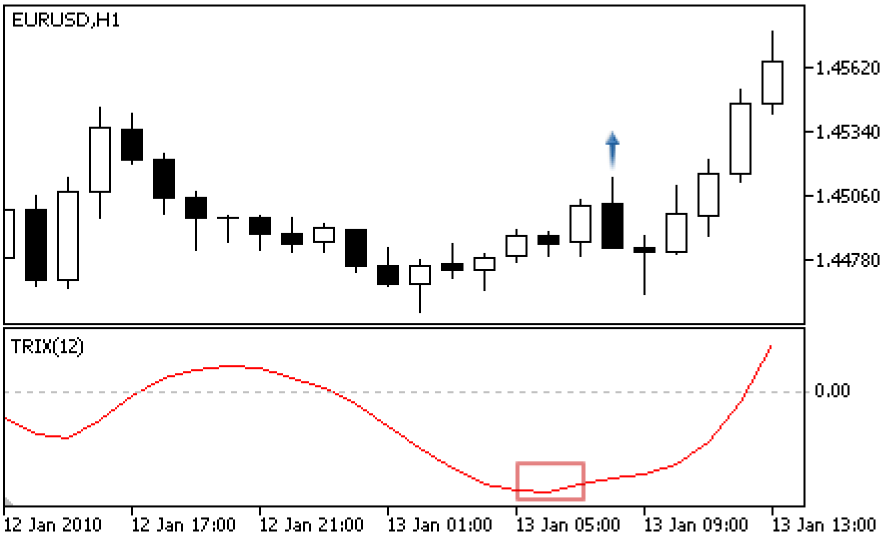
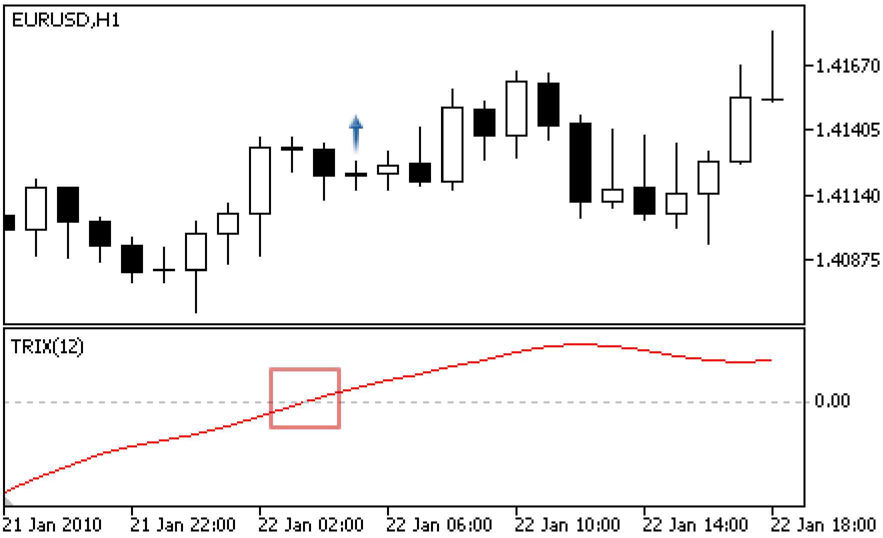
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodK	Period of calculation of the main line of the oscillator.
PeriodD	Period of averaging of the main line of the oscillator.
PeriodSlow	Period of slowing.
Applied	A price series used for calculation of the oscillator.

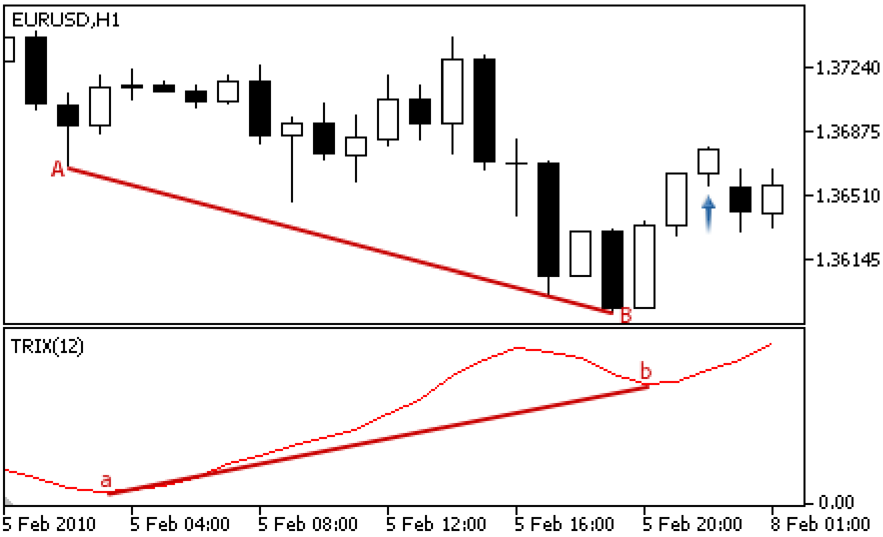
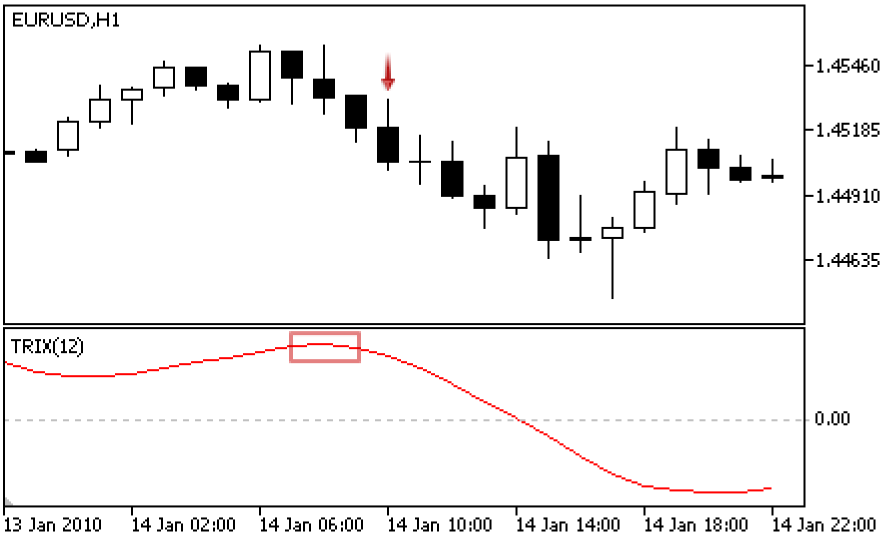
Signals of the Oscillator Triple Exponential Average

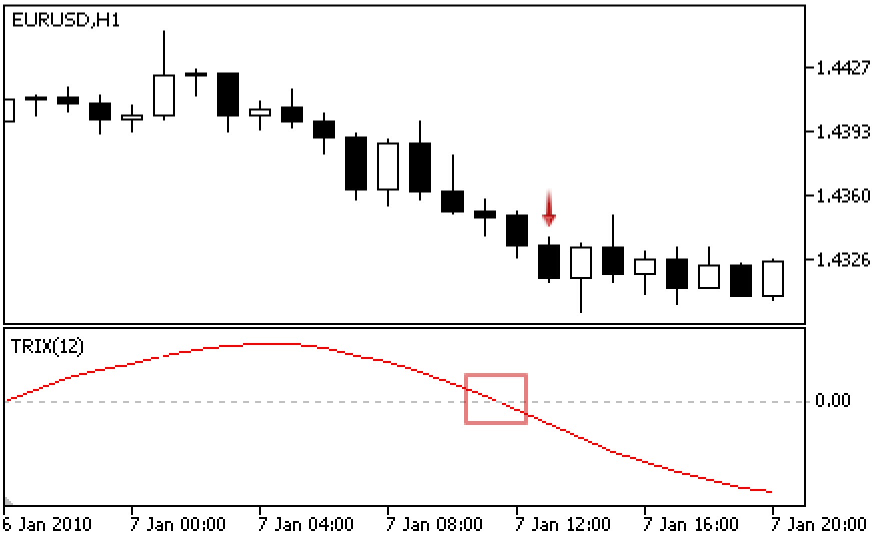
This module of signals is based on the market models of the oscillator Triple Exponential Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

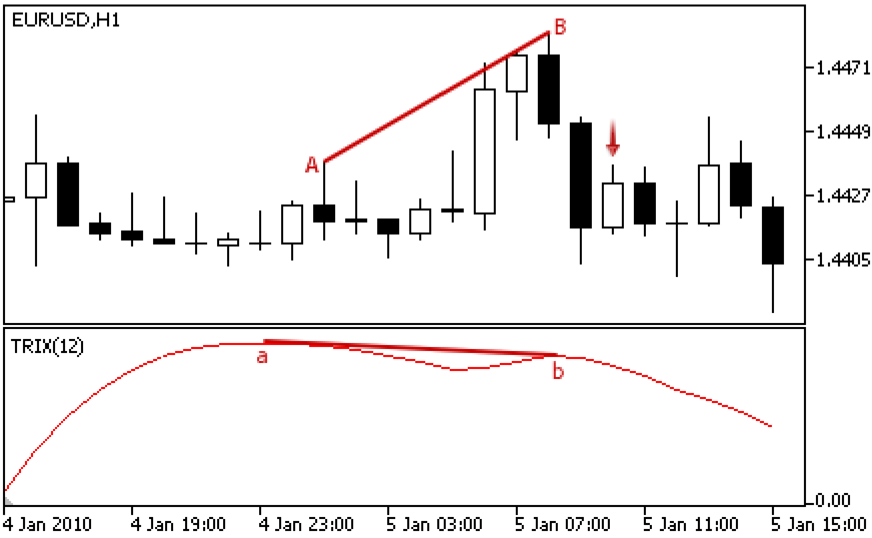
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	- **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  - **Crossing the zero level** — the main line is above the zero level at the analyzed bar and below the zero level at the previous one. 

Signal Type	Description of Conditions
	• **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 
For selling	• **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  • **Crossing the zero level** — the main line is below the zero level at the analyzed bar and above the zero level at the previous one.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td>


- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.



</td></tr>
<tr><td>No objections to buying</td><td>Value of the oscillator grows at the analyzed bar.</td></tr>
<tr><td>No objections to selling</td><td>Value of the oscillator falls at the analyzed bar.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

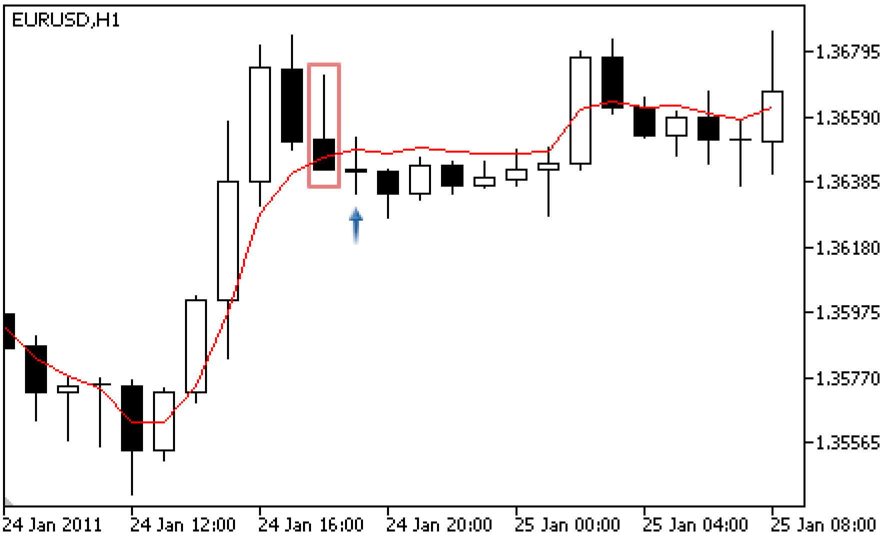
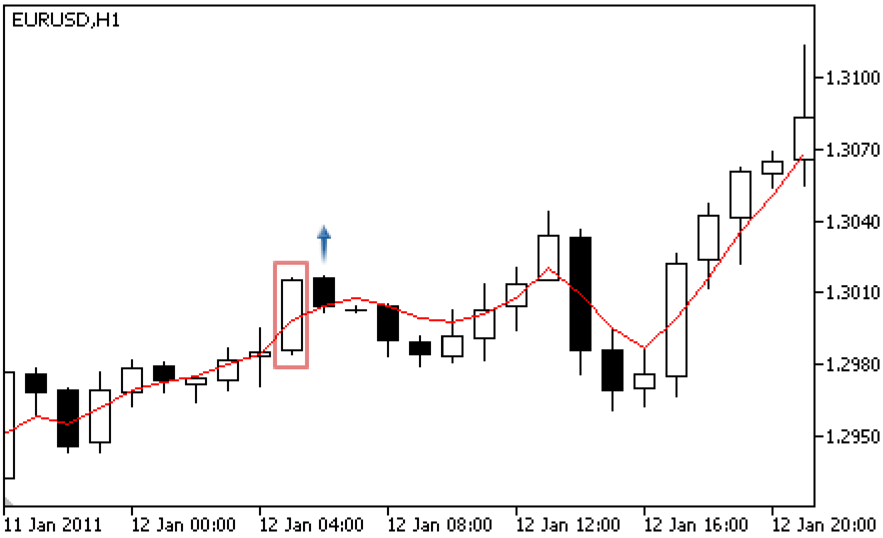
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodTriX	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

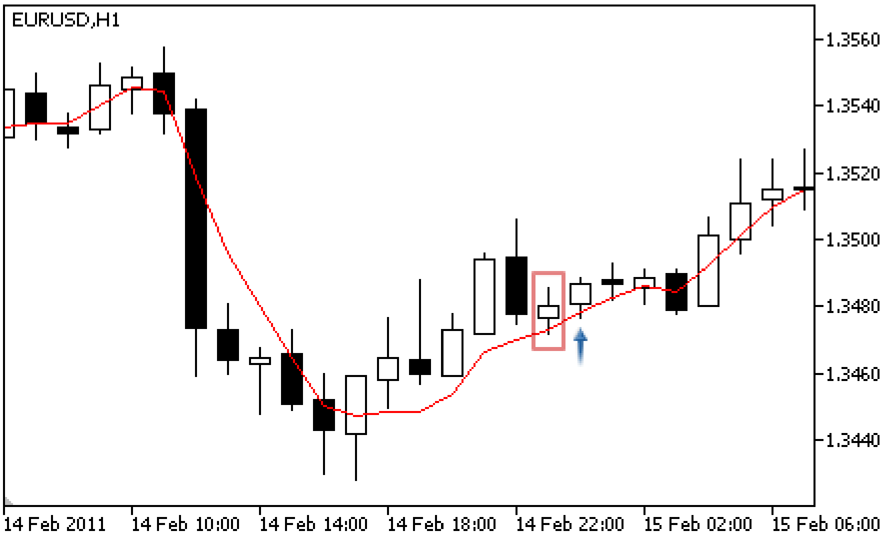
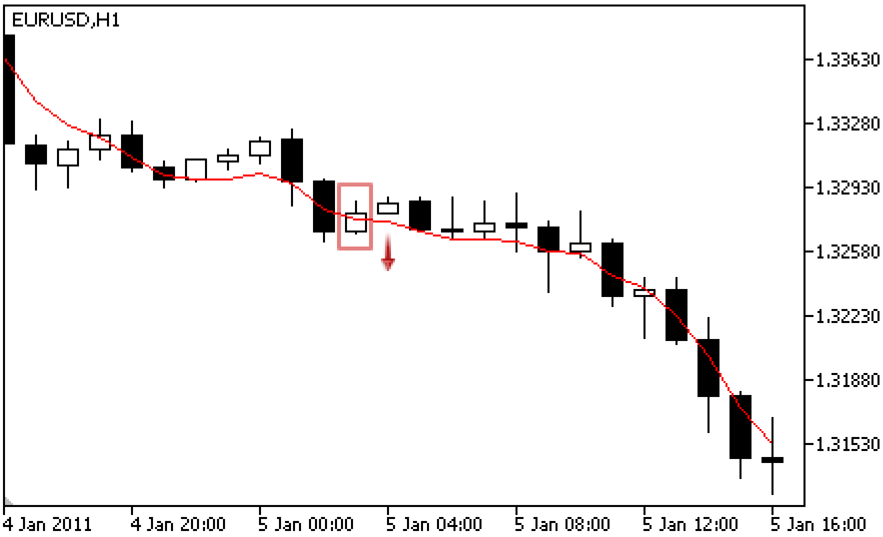
Signals of the Indicator Triple Exponential Moving Average

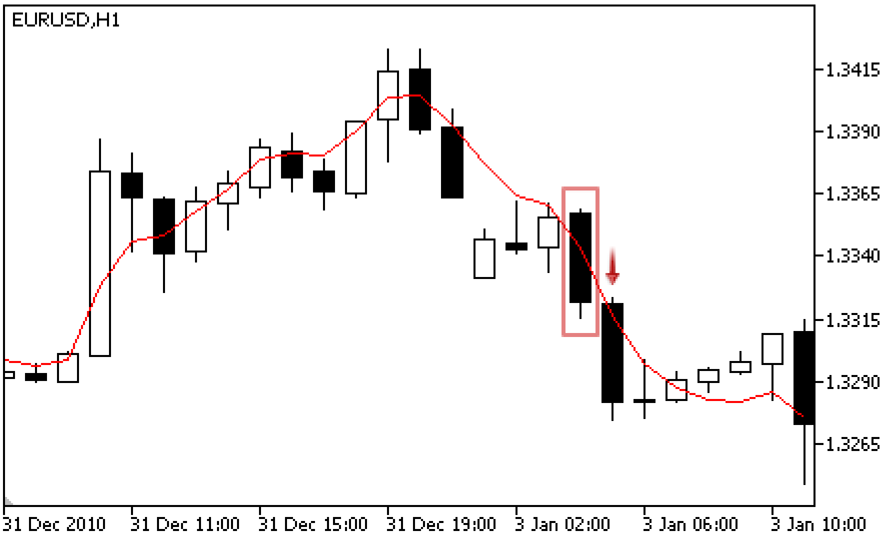
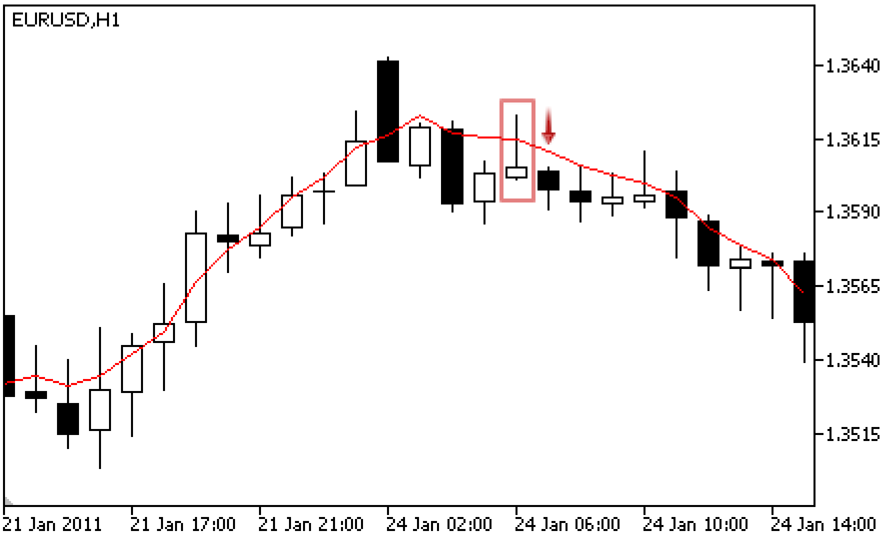
This module of signals is based on the market models of the indicator Triple Exponential Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal of a roll-back from the indicator line).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	- Formed breakout. The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal of a roll-back from the indicator line). 
For selling	- **Failed breakout.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal of a roll-back from the indicator line).  - **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 • **Formed breakout.** The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar are below the indicator, and the High price is above the indicator) and the indicator falls (weak signal of a roll-back from the indicator line). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shift of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Oscillator Williams Percent Range

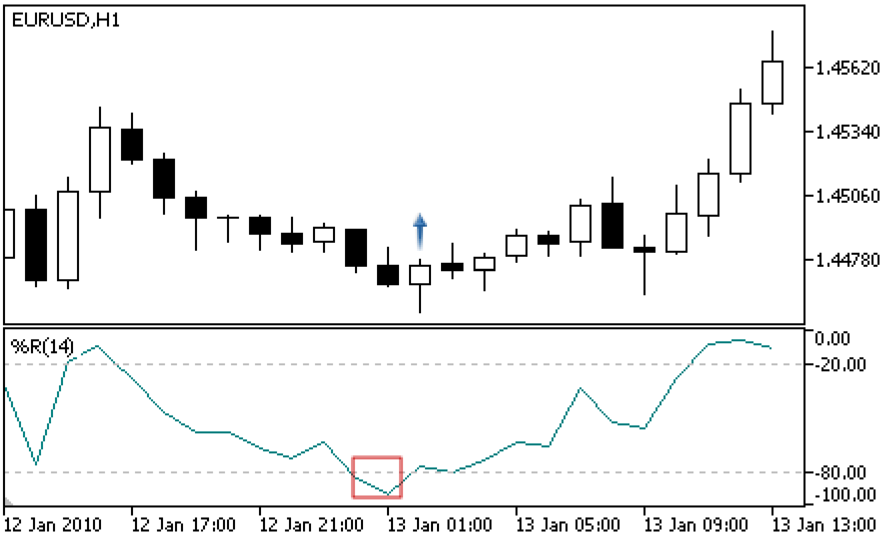
This module of signals is based on the market models of the oscillator Williams Percent Range. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

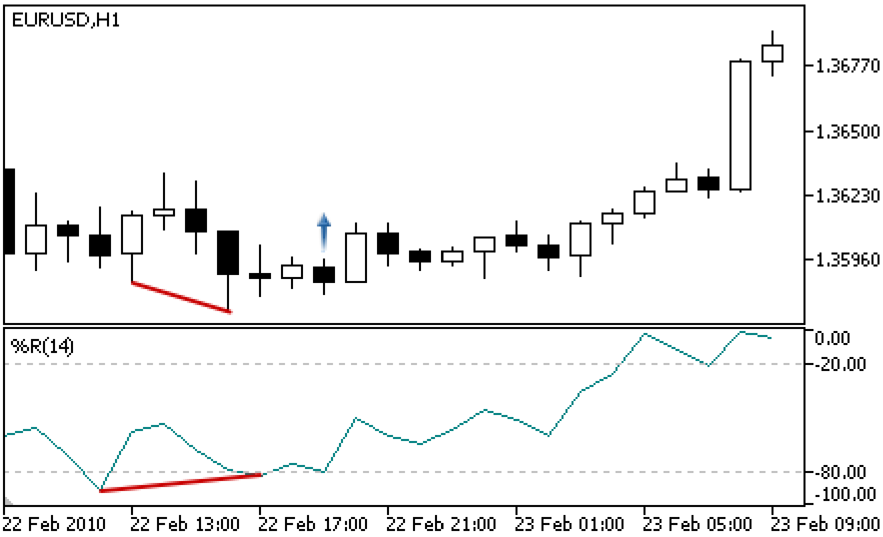
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td>For buying</td><td>

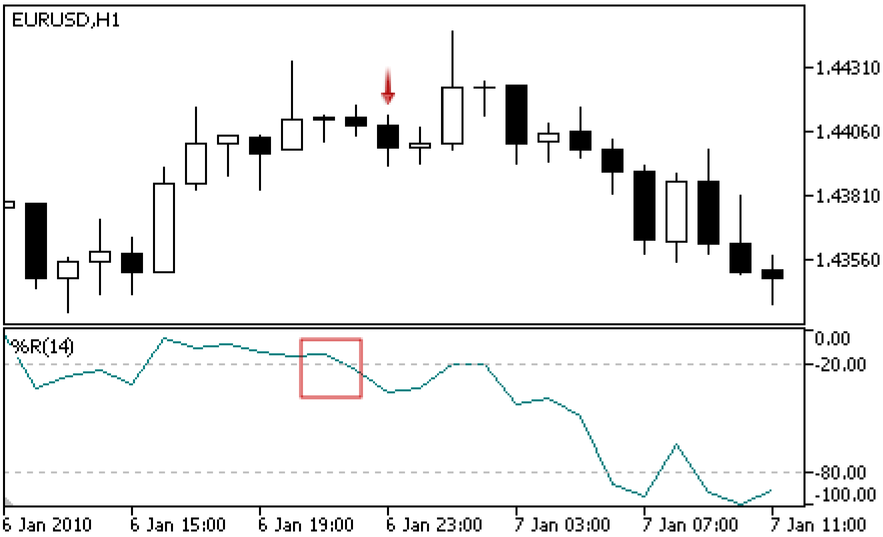
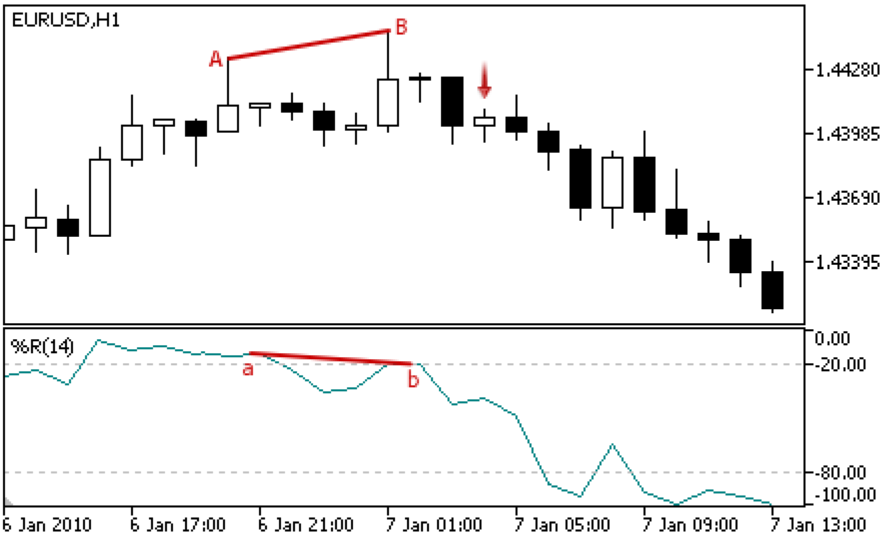
- **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is -80).



- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.



</td></tr>
</table>

Signal Type	Description of Conditions
For selling	- **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is -20).  - **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Remember that the oscillator Williams Percent Range has a reversed scale. Its maximum value is -100, and minimum is 0.

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodWPR	Period of calculation of the oscillator.

Trailing Stop classes

This section contains technical details of working with trailing stop classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert\Trailing folder.

Class	Description
CTrailingFixedPips	This class implements Trailing Stop algorithm based on fixed points
CTrailingMA	This class implements Trailing Stop algorithm based on the values of Moving Average indicator
CTrailingNone	A stub class, it does not uses any Trailing Stop algorithm
CTrailingPSAR	This class implements Trailing Stop algorithm based on the values of Parabolic SAR indicator

CTrailingFixedPips

CTrailingFixedPips is a class with implementation of Trailing Stop algorithm based on fixed points trailing.

CTrailingFixedPips class implements the following algorithm of trailing open positions: If Stop Loss level is equal to zero, Stop Loss order modification condition is considered unfulfilled, therefore there is no suggestion to change a position's Stop Loss. Otherwise, the check of whether the price has moved in favorable direction is performed.

If a position has a Stop Loss order, it checks the minimal allowed Stop Loss distance to the current price. If the position has no Stop Loss level, the distance between the current and open prices is checked. If the distance exceeds Stop Loss level, the system suggests to set a new Stop Loss price.

If Stop Loss modification condition is fulfilled and Take Profit is not equal to zero, the system suggests setting a new Take Profit price.

If Expert Advisor class has been initialized with the flag every_tick=false, it will perform all operations (trading, trailing, etc) only at the new bar on a working symbol and timeframe. In this case, setting Take Profit order allows you to close position when the price moves in the position direction before a new bar appears.

Description

CTrailingFixedPips implements the Trailing Stop algorithm at a specified "distance" from the current price (in points).

Declaration

```
class CTrailingFixedPips: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\CTrailingFixedPips.mqh>
```

Inheritance hierarchy

CObject
  CExpertBase
    CExpertTrailing
      CTrailingFixedPips

Class Methods by Groups

Initialization	
StopLevel	Sets the value of Stop Loss level
ProfitLevel	Sets the value of Take Profit level
virtual ValidationSettings	Checks the settings
Check Trailing Methods	

Initialization	
virtual CheckTrailingStopLong	Check Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

StopLevel

Sets the value of Stop Loss level.

```
void  StopLevel(
   int   stop_level       // level
   )
```

Parameters

stop_loss

[in] The value of Stop Loss level (in conventional 2/4-digit points).

Note

If Stop Loss level is equal to 0, the Trailing Stop is not used.

ProfitLevel

Sets the value of Take Profit level.

```
void  ProfitLevel(
   int  profit_level      // level
   )
```

Parameters

profit_level

[in] The value of Take Profit level (in conventional 2/4-digit points).

Note

If profit level is equal to 0, the Trailing Stop is not used.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function checks Take Profit and Stop Loss levels. The correct values are 0 and values greater than the minimal indention in points from the current close price to place Stop orders.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // pointer
   double&         sl,            // reference to Stop Loss
   double&         tp             // reference to Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

If Stop Loss level is equal to 0, the condition is not fulfilled (exit). If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the current Bid price is higher than base price+stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Bid price+take profit level.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,      // pointer
   double&         sl,            // reference to Stop Loss
   double&         tp             // reference to Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

If Stop Loss level is equal to 0, the condition is not fulfilled (exit). If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the current Ask price is lower than base price - stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Ask price - take profit level.

CTrailingMA

CTrailingMA is a class with implementation of Trailing Stop algorithm based on the values of moving average indicator.

Description

CTrailingMA class implements Trailing Stop algorithm based on the values of moving average indicator of the previous (completed) bar.

Declaration

```
class CTrailingMA: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingMA.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertTrailing
      - CTrailingMA

Class Methods by Groups

Initialization	
Period	Sets period of moving average
Shift	Sets shift of moving average
Method	Sets smoothing method of moving average
Applied	Sets applied price of moving average
virtual InitIndicators	Initializes indicators and timeseries
virtual ValidationSettings	Checks the settings
Check Trailing Methods	
virtual CheckTrailingStopLong	Check Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries

Period

Sets period of a moving average.

```
void  Period(
   int  period      // period
   )
```

Parameters

period

[in] Period of a moving average.

Shift

Sets shift of a moving average.

```
void  Shift(
   int   shift       // parameter
   )
```

Parameters

shift

[in] Shift of a moving average.

Method

Sets smoothing method of a moving average.

```
void  Method(
   ENUM_MA_METHOD  method      // parameter
   )
```

Parameters

method

[in] Smoothing method (from ENUM_MA_METHOD enumeration) of a moving average.

Applied

Sets applied price of a moving average.

```
void  Applied(
   ENUM_APPLIED_PRICE  applied      // parameter
   )
```

Parameters

applied

[in] Applied price (from ENUM_APPLIED_PRICE enumeration) of a moving average.

InitIndicators

Initializes indicators and timeseries.

```
virtual bool  InitIndicators(
   CIndicators*   indicators      // pointer
   )
```

Parameters

indicators

[in] Pointer to indicators and timeseries collection (CExpert class member).

Return Value

true - successful, otherwise - false.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function checks the period of moving average, the correct values are positive.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // pointer
   double&         sl,           // reference
   double&         tp            // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the maximal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position. If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,      // pointer
   double&         sl,            // reference
   double&         tp             // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the minimal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position. If the calculated Stop Loss price is lower than base price and higher than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

CTrailingNone

CTrailingNone is a stub class. This class should be used at initialization of trailing object in CExpert class if your strategy does not use Trailing Stop.

Description

CTrailingNone class does not implement any Trailing Stop algorithm. The methods of checking Trailing Stop conditions always return false.

Declaration

```
class CTrailingNone: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingNone.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertTrailing
      - CTrailingNone

Class Methods by Groups

Check Trailing Methods	
virtual CheckTrailingStopLong	A stub method for checking Trailing Stop conditions of a long position
virtual CheckTrailingStopShort	A stub method for checking Trailing Stop conditions of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertTrailing

CheckTrailingStopLong, CheckTrailingStopShort

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,      // pointer
   double&         sl,            // reference
   double&         tp             // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

The function always returns false.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,      // pointer
   double&         sl,            // reference
   double&         tp             // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

The function always returns false.

CTrailingPSAR

CTrailingPSAR is a class with implementation of Trailing Stop algorithm based on the values of Parabolic SAR indicator.

Description

CTrailingPSAR class implements the Trailing Stop algorithm based on the values of Parabolic SAR indicator of the previous (completed) bar.

Declaration

```
class CTrailingPSAR: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingParabolicSAR.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertTrailing
      - CTrailingPSAR

Class Methods by Groups

Initialization	
Step	Sets the value of step of Parabolic SAR indicator
Maximum	Sets the value of maximum of Parabolic SAR indicator
virtual InitIndicators	Initializes indicators and timeseries
Check Trailing Methods	
virtual CheckTrailingStopLong	Check conditions of trailing stop of a long position
virtual CheckTrailingStopShort	Check conditions of trailing stop of a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, ValidationSettings, SetPriceSeries, SetOtherSeries

Step

Sets the value of step of Parabolic SAR indicator.

```
void  Step(
   double  step      // parameter
   )
```

Parameters

step

[in] The value of step of Parabolic SAR indicator.

Maximum

Sets the value of maximum of Parabolic SAR indicator.

```
void  Maximum(
   double  maximum      // parameter
   )
```

Parameters

maximum

[in] The value of maximum of Parabolic SAR indicator.

InitIndicators

Initializes indicators and timeseries.

```
virtual bool  InitIndicators(
   CIndicators*   indicators      // pointer
   )
```

Parameters

indicators

[in] Pointer to indicators and timeseries collection (CExpert class member).

Return Value

true - successful, otherwise - false.

CheckTrailingStopLong

Checks Trailing Stop conditions of a long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*   position,      // pointer
   double&          sl,            // reference
   double&          tp             // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the maximal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of a short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*   position,      // pointer
   double&          sl,            // reference
   double&          tp             // reference
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Reference to variable for Stop Loss price.

tp

[in][out] Reference to variable for Take Profit price.

Return Value

true - conditions are satisfied, otherwise - false.

Note

First it calculates the minimal allowed Stop Loss price closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price. If the calculated Stop Loss price is lower than base price and higher than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

Money Management classes

This section contains technical details of working with money and risk management classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) trading strategies.

MQL5 Standard Library (in terms of money and risk management classes) is placed in the terminal directory, in the Include\Expert\Money\ folder.

Class	Description
CMoneyFixedLot	This class implements money management algorithm based on trading with predefined fixed lot size.
CMoneyFixedMargin	This class implements money management algorithm based on trading with predefined fixed margin.
CMoneyFixedRisk	This class implements money management algorithm based on trading with predefined risk.
CMoneyNone	This class implements money management algorithm based on trading with minimal allowed lot size.
CMoneySizeOptimized	This class implements money management algorithm based on trading with variable lot size, depending on results of the previous deals.

CMoneyFixedLot

CMoneyFixedLot is the class designed to implement money management algorithm based on trading with predefined fixed lot size.

Description

CMoneyFixedLot implements money management algorithm based on trading with predefined fixed lot size.

Declaration

```
class CMoneyFixedLot: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedLot.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertMoney
      - CMoneyFixedLot

Class Methods by Groups

Initialization	
Lots	Sets trading volume
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertMoney

Percent, CheckReverse, CheckClose

Lots

Sets trading volume (in lots).

```
void  Lots(
   double   lots       // number of lots
   )
```

Parameters

lots

[in] Trading volume.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

Checks if the predefined fixed lot falls within the range of allowed values and is multiple of change step.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double  CheckOpenLong(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss order price.

Return Value

Trade volume for a long position.

Note

The function always returns the predefined fixed trade volume.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double  CheckOpenShort(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss order price.

Return Value

Trade volume for a short position.

Note

The function always returns the predefined fixed trade volume.

CMoneyFixedMargin

CMoneyFixedMargin is the class designed to implement money management algorithm based on trading with predefined fixed margin.

Description

CMoneyFixedMargin implements money management algorithm based on trading with predefined fixed margin.

Declaration

```
class CMoneyFixedMargin: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedMargin.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertMoney
      - CMoneyFixedMargin

Class Methods by Groups

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertMoney

Percent, ValidationSettings, CheckReverse, CheckClose

CheckOpenLong

Gets trade volume for a long position.

```
virtual double  CheckOpenLong(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a long position with a fixed predefined margin. The margin is defined by Percent parameter of CExpertMoney base class.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double  CheckOpenShort(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function returns trade volume for a short position with a predefined fixed margin. The margin is defined by Percent parameter of CExpertMoney base class.

CMoneyFixedRisk

CMoneyFixedRisk is a class with implementation of money management algorithm with fixed predefined risk.

Description

CMoneyFixedRisk class implements the money management algorithm with fixed predefined risk.

Declaration

```
   class CMoneyFixedRisk: public CExpertMoney
```

Title

```
   #include <Expert\Money\MoneyFixedRisk.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertMoney
      - CMoneyFixedRisk

Class Methods by Groups

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertMoney

Percent, ValidationSettings, CheckReverse

CheckOpenLong

Gets trade volume for a long position.

```
virtual double  CheckOpenLong(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a long position with a fixed predefined risk level. The risk is defined by Percent parameter of CExpertMoney base class.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double  CheckOpenShort(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function returns trade volume for a short position with a fixed predefined risk level. The risk is defined by Percent parameter of CExpertMoney base class.

CMoneyNone

CMoneyNone is a class with implementation of the "absence" of money and risk management.

Description

CMoneyNone class implements trading with minimal allowed lot.

Declaration

```
class CMoneyNone: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyNone.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertMoney
      - CMoneyNone

Class Methods by Groups

Initialization	
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertMoney

Percent, CheckReverse, CheckClose

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

The function always returns true.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double  CheckOpenLong(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function always returns the minimal lot size.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double  CheckOpenShort(
   double  price,      // price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a short position.

Note

The function always returns the minimal lot size.

CMoneySizeOptimized

CMoneySizeOptimized is a class with implementation of money and risk management algorithm depending on results of the previous deals.

Description

CMoneySizeOptimized implements the market entry algorithm with the lot size depending on results of the previous deals.

Declaration

```
   class CMoneySizeOptimized: public CExpertMoney
```

Title

```
   #include <Expert\Money\MoneySizeOptimized.mqh>
```

Inheritance hierarchy

- CObject
  - CExpertBase
    - CExpertMoney
      - CMoneySizeOptimized

Class Methods by Groups

Initialization	
DecreaseFactor	Sets the parameter value
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for a long position
virtual CheckOpenShort	Gets trade volume for a short position

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CExpertBase

InitPhase, TrendType, UsedSeries, EveryTick, Open, High, Low, Close, Spread, Time, TickVolume, RealVolume, Init, Symbol, Period, Magic, SetMarginMode, SetPriceSeries, SetOtherSeries, InitIndicators

Methods inherited from class CExpertMoney

Percent, CheckReverse, CheckClose

DecreaseFactor

Sets the value of decrease factor.

```
void  DecreaseFactor(
   double  decrease_factor      // decrease factor
   )
```

Parameters

decrease_factor

[in] Decrease factor.

Note

The DecreaseFactor defines the open position volume decreasing coefficient (compared with the volume of a previous position) for the case of consecutive loss trades.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Return Value

true - successful, otherwise - false.

Note

If the value of decrease factor is negative, it returns false, otherwise it returns true.

CheckOpenLong

Gets trade volume for a long position.

```
virtual double  CheckOpenLong(
   double   price,      // price
   double   sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for long position.

Note

The function returns trade volume for a long position considering results of previous deals.

CheckOpenShort

Gets trade volume for a short position.

```
virtual double  CheckOpenShort(
   double   price,      // price
   double   sl          // Stop Loss price
   )
```

Parameters

price

[in] Estimated open price.

sl

[in] Estimated Stop Loss price.

Return Value

Trade volume for a long position.

Note

The function returns trade volume for a short position considering results of previous deals.

Classes for Creating Control Panels and Dialogs

This section contains technical details of working with classes for creation of controls panels and dialogs, as well as description of the relevant components of the MQL5 standard library.

The use of these classes will save time when developing custom interactive MQL5 applications, including Expert Advisors and indicators.

MQL5 Standard Library (in terms of classes for creation of control panels and dialogs) is placed in the terminal data folder, in the MQL5\Include\Controls.

Find the examples of working with classes in the following articles:

- How to create a graphical panel of any complexity level
- Improving Panels: Adding transparency, changing background color and inheriting from CAppDialog/CWndClient
- Adding a control panel to an indicator or an Expert Advisor in no time
- Create your own graphical panels in MQL5
- Creating active control panels in MQL5 for trading

The sample Expert Advisor, which illustrates the operation of these classes, can be found in MQL5\Expert\Examples\Controls.

Auxiliary structures	Description
CRect	Structure of the rectangular area
CDateTime	Structure for working with date and time

Base classes	Description
CWnd	Base class for all controls
CWndObj	Base class for controls and dialogs
CWndContainer	Base class for complex controls

Simple controls	Description
CLabel	Control, based on "Text label" graphic object
CBmpButton	Control, based on "Bitmap label" graphic object
CButton	Control, based on "Button" graphic object
CEdit	Control, based on "Edit field" graphic object
CPanel	Control, based on "Rectangle label"
CPicture	Control, based on "Bitmap label"

Complex controls	Description
CScroll	Base class of the scroll bar
CScrollV	Vertical scroll bar
CScrollH	Horizontal scroll bar
CWndClient	Base class of the client area with scroll bars
CListView	List view
CComboBox	Combo box
CCheckBox	Check box
CCheckGroup	Check group
CRadioButton	Radio button
CRadioGroup	Radio group
CSpinEdit	Increment/decrement field
CDialog	Dialog
CAppDialog	Main dialog of MQL5 application

CRect

CRect is a class of the rectangular area of the chart.

Description

CRect is a class of the area, it is defined by both coordinates of the upper-left and lower-right corners of a rectangle in Cartesian coordinates.

Declaration

```
class CRect
```

Title

```
#include <Controls\Rect.mqh>
```

Class Methods by Groups

Properties	
Left	Gets/sets the X coordinate of the upper-left corner
Top	Gets/sets the Y coordinate of the upper-left corner
Right	Gets/sets the X coordinate of the lower-right corner
Bottom	Gets/sets the Y coordinate of the lower-right corner
Width	Gets/sets the width
Height	Gets/sets the height
SetBound	Sets new coordinates of the area
Move	Performs the absolute displacement of the area coordinates
Shift	Performs the relative displacement (shift) of the area coordinates
Contains	Checks if the point is inside the area
Additional methods	
Format	Gets the area coordinates as a string

Left (Get Method)

Gets the X coordinate of the upper-left corner.

```
int  Left()
```

Return Value

X coordinate of the upper-left corner.

Left (Set Method)

Sets the X coordinate of the upper-left corner.

```
void  Left(
   const int   x        // x coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Return Value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner.

```
int  Top()
```

Return Value

Y coordinate of the upper-left corner.

Top (Set Method)

Sets the Y coordinate of the upper-left corner.

```
void  Top(
   const int  y       // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Return Value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner.

```
int  Right()
```

Return Value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the X coordinate of the lower-right corner.

```
void  Right(
   const int  x      // x coordinate
   )
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Return Value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner.

```
int  Bottom()
```

Return Value

Y coordinate of the lower-right corner.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner.

```
void  Bottom(
   const int  y       // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Return Value

None.

Width (Get Method)

Gets the width of the area.

```
int  Width()
```

Return Value

Width of the area.

Width (Set Method)

Sets new width of the area.

```
virtual bool  Width(
   const int  w       // width
   )
```

Parameters

w

[in] New width.

Return Value

true - successful, otherwise - false.

Height (Get Method)

Gets the height of the area.

```
int  Height()
```

Return Value

Height of the area.

Height (Set Method)

Sets new height of the area.

```
virtual bool  Height(
   const int  h       // height
   )
```

Parameters

h

[in] New height.

Return Value

true - successful, otherwise - false.

SetBound

Sets new coordinates of the area using CRect class coordinates.

```
void  SetBound(
   const & CRect   rect       // CRect class
   )
```

Return Value

None.

SetBound

Sets new coordinates of the area.

```
void  SetBound(
   const int  l       // left
   const int  t       // top
   const int  r       // right
   const int  b       // bottom
   )
```

Parameters

l

[in] X coordinate of the upper-left corner.

t

[in] Y coordinate of the upper-left corner.

r

[in] X coordinate of the lower-right corner.

b

[in] Y coordinate of the lower-right corner.

Return Value

None.

Move

Performs the absolute displacement of the area coordinates.

```
void  Move(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] New X coordinate.

y

[in] New Y coordinate.

Return Value

None.

Shift

Performs the relative displacement (shift) of the area coordinates.

```
void  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

None.

Contains

Checks if the point is inside the CRect class area.

```
bool  Contains(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Return Value

true - the point is inside the area (including borders), otherwise - false.

Format

Gets the area coordinates as a string.

```
string  Format(
   string &  fmt,     // format
   ) const
```

Parameters

fmt

[in] String with format.

Return Value

String with the area coordinates.

CDateTime

CDateTime is a structure for working with date and time.

Description

CDateTime is a structure derived from MqlDateTime, it used for operation with date and time in controls.

Declaration

```
struct CDateTime
```

Title

```
#include <Tools\DateTime.mqh>
```

Methods

Properties	
MonthName	Gets month name
ShortMonthName	Gets short name of the month
DayName	Gets full name of the day in a week
ShortDayName	Gets short name of the day in a week
DaysInMonth	Gets number of days in month
Get/Set methods	
DateTime	Gets/sets date and time
Date	Sets date
Time	Sets time
Sec	Sets seconds
Min	Sets minutes
Hour	Sets hour
Day	Sets day of the month
Mon	Sets month
Year	Sets year
Additional methods	
SecDec	Subtracts specified number of seconds
SecInc	Adds specified number of seconds
MinDec	Subtracts specified number of minutes

Properties	
MinInc	Adds specified number of minutes
HourDec	Subtracts specified number of hours
HourInc	Adds specified number of hours
DayDec	Subtracts specified number of days
DayInc	Adds specified number of days
MonDec	Subtracts specified number of months
MonInc	Adds specified number of months
YearDec	Subtracts specified number of years
YearInc	Adds specified number of years

MonthName

Gets month name.

```
string  MonthName()  const
```

Gets month name by index.

```
string  MonthName(
   const int      num          // month index
   )  const
```

Parameters

num

[in] Month index (1-12).

Return Value

Full month name.

ShortMonthName

Gets short month name.

```
string  ShortMonthName()  const
```

Gets short month name by index.

```
string  ShortMonthName(
   const int      num          // month index
   )  const
```

Parameters

num

[in] Month index (1-12).

Return Value

Short month name.

DayName

Gets full name of the day in a week.

```
string  DayName()  const
```

Gets full name of the day in a week by index.

```
string  DayName(
   const int      num          // day index
   )  const
```

Parameters

num

[in] Day index (0-6).

Return Value

Full name of the day.

ShortDayName

Gets short name of the day in a week.

```
string  ShortDayName()  const
```

Gets short name of the day in a week by index.

```
string  ShortDayName(
   const int      num         // day index
   )  const
```

Parameters

num

[in] Day index (0-6).

Return Value

Short name of the day.

DaysInMonth

Gets number of days in month.

```
int  DaysInMonth()  const
```

Return Value

Number of days in month.

DateTime (Get Method)

Gets date and time.

```
datetime  DateTime()
```

Return Value

Value of datetime type.

DateTime (Set Method datetime)

Sets date and time with datetime type.

```
void  DateTime(
   const datetime       value          // date and time
   )
```

Parameters

value

[in] Value of datetime type.

Return Value

None.

DateTime (Set Method MqlDateTime)

Sets date and time with MqlDateTime type.

```
void  DateTime(
   const MqlDateTime    &value         // date and time
   )
```

Parameters

value

[in] Value of MqlDateTime type.

Return Value

None.

Date (Set Method datetime)

Sets date with datetime type.

```
void  Date(
   const datetime       value          // date
   )
```

Parameters

value

[in] Value of datetime type.

Return Value

None.

Date (Set Method MqlDateTime)

Sets date with MqlDateTime type.

```
void  Date(
   const MqlDateTime    &value         // date
   )
```

Parameters

value

[in] Value of MqlDateTime type.

Return Value

None.

Time (Set Method datetime)

Sets time with datetime type.

```
void  Time(
   const datetime       value          // time
   )
```

Parameters

value

[in] Value of datetime type.

Return Value

None.

Time (Set Method MqlDateTime)

Sets time with MqlDateTime type.

```
void  Time(
   const MqlDateTime    &value         // time
   )
```

Parameters

value

[in] Value of MqlDateTime type.

Return Value

None.

Sec

Sets seconds.

```
void  Sec(
   const int    value          // seconds
   )
```

Parameters

value

[in] Seconds.

Return Value

None.

Min

Sets minutes.

```
void  Min(
   const int     value          // minutes
   )
```

Parameters

value

[in] Minutes.

Return Value

None.

Hour

Sets hour.

```
void  Hour(
   const int    value          // hour
   )
```

Parameters

value

[in] Hour.

Return Value

None.

Day

Sets day of the month.

```
void  Day(
   const int    value          // day of month
   )
```

Parameters

value

[in] Day of month.

Return Value

None.

Mon

Sets month.

```
void  Mon(
   const int    value          // month
   )
```

Parameters

value

[in] Month.

Return Value

None.

Year

Sets year.

```
void  Year(
   const int    value         // year
   )
```

Parameters

value

[in] Year.

Return Value

None.

SecDec

Subtracts specified number of seconds.

```
void  SecDec(
   int    delta=1         // seconds
   )
```

Parameters

delta

[in] Seconds to subtract.

Return Value

None.

SecInc

Adds specified number of seconds.

```
void  SecInc(
   int    delta=1         // seconds
   )
```

Parameters

delta

[in] Seconds to add.

Return Value

None.

MinDec

Subtracts specified number of minutes.

```
void  MinDec(
   int    delta=1         // minutes
   )
```

Parameters

delta

[in] Minutes to subtract.

Return Value

None.

MinInc

Adds specified number of minutes.

```
void  MinInc(
   int    delta=1         // minutes
   )
```

Parameters

delta

[in] Minutes to add.

Return Value

None.

HourDec

Subtracts specified number of hours.

```
void  HourDec(
   int    delta=1          // hours
   )
```

Parameters

delta

[in] Hours to subtract.

Return Value

None.

HourInc

Adds specified number of hours.

```
void  HourInc(
   int    delta=1         // hours
   )
```

Parameters

delta

[in] Hours to add.

Return Value

None.

DayDec

Subtracts specified number of days.

```
void  DayDec(
   int    delta=1         // days
   )
```

Parameters

delta

[in] Days to subtract.

Return Value

None.

DayInc

Adds specified number of days.

```
void  DayInc(
   int    delta=1         // days
   )
```

Parameters

delta

[in] Days to add.

Return Value

None.

MonDec

Subtracts specified number of months.

```
void  MonDec(
   int    delta=1         // months
   )
```

Parameters

delta

[in] Months to subtract.

Return Value

None.

MonInc

Adds specified number of months.

```
void  MonInc(
   int    delta=1         // months
   )
```

Parameters

delta

[in] Months to add.

Return Value

None.

YearDec

Subtracts specified number of years.

```
void  YearDec(
   int    delta=1         // years
   )
```

Parameters

delta

[in] Years to subtract.

Return Value

None.

YearInc

Adds specified number of years.

```
void  YearInc(
   int    delta=1         // years
   )
```

Parameters

delta

[in] Years to add.

Return Value

None.

CWnd

CWnd is a base class for all Standard Library controls.

Description

CWnd class is the implementation of the base control class.

Declaration

```
class CWnd : public CObject
```

Title

```
#include <Controls\Wnd.mqh>
```

Inheritance hierarchy

CObject
    CWnd

Direct descendants

CDragWnd, CWndContainer, CWndObj

Class Methods by Groups

Create and destroy	
Create	Creates control
Destroy	Destroys control
Chart event handlers	
OnEvent	Event handler of all chart events
OnMouseEvent	Event handler for the CHARTEVENT_MOUSE_MOVE event
Name	
Name	Gets control name
Access to container	
ControlsTotal	Gets the number of controls in the container
Control	Gets the control by index
ControlFind	Gets the control by ID
Geometry	
Rect	Gets the control coordinates
Left	Gets/sets the X coordinate of the upper-left corner

Create and destroy	
Top	Gets/sets the Y coordinate of the upper-left corner
Right	Gets/sets the X coordinate of the lower-right corner
Bottom	Gets/sets the Y coordinate of the lower-right corner
Width	Gets/sets the control width
Height	Gets/sets the control height
Move	Control absolute displacement
Shift	Control relative displacement
Resize	Changes control size
Contains	Checks if the point/control is inside the control area
Align	
Alignment	Sets alignment properties of the control
Align	Performs control alignment in the specified chart area
Identification	
Id	Gets/sets the control ID
State	
IsEnabled	Checks the control ability to respond to user's actions
Enable	Enables the control ability to respond to user's actions
Disable	Disables the control ability to respond to user's actions
IsVisible	Checks the visibility flag
Visible	Sets the visibility flag
Show	Shows the control
Hide	Hides the control
IsActive	Checks the control activity
Activate	Activates the control
Deactivate	Deactivates the control
State flags	
StateFlags	Gets/sets the control state flags
StateFlagsSet	Sets the control state flags
StateFlagsReset	Resets the control state flags
Properties flags	

Create and destroy	
PropFlags	Gets/sets the control properties flags
PropFlagsSet	Sets the control properties flags
PropFlagsReset	Resets the control properties flags
Mouse operations	
MouseX	Gets/saves the mouse X coordinate
MouseY	Gets/saves the mouse Y coordinate
MouseFlags	Gets/saves the mouse buttons state
MouseFocusKill	Kills mouse focus
Internal event handlers	
OnCreate	"Create" event handler
OnDestroy	"Destroy" event handler
OnMove	"Move" event handler
OnResize	"Resize" event handler
OnEnable	"Enable" event handler
OnDisable	"Disable" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler
OnClick	"Click" event handler
OnChange	"Change" event handler
Mouse event handlers	
OnMouseDown	"MouseDown" event handler
OnMouseUp	"MouseUp" event handler
Drag event handlers	
OnDragStart	"DragStart" event handler
OnDragProcess	"DragProcess" event handler
OnDragEnd	"DragEnd" event handler
Drag object	
DragObjectCreate	Creates drag object

Create and destroy	
DragObjectDestroy	Destroys drag object

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Create

Creates a control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow, at which the control is created.

x1

[in] X coordinate of the upper-left corner.

y1

[in] Y coordinate of the upper-left corner.

x2

[in] X coordinate of the lower-right corner.

y2

[in] Y coordinate of the lower-right corner.

Return Value

true - successful, otherwise - false.

Note

Base class method only saves the creation parameters and always returns true.

Destroy

Destroys a control.

```
virtual bool  Destroy()
```

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // event parameter
   const double&    dparam,      // event parameter
   const string&    sparam       // event parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Return Value

true - event has been processed, otherwise - false.

OnMouseEvent

Mouse event handler (the CHARTEVENT_MOUSE_MOVE chart event).

```
virtual bool  OnMouseEvent(
   const int  x,          // x coordinate
   const int  y,          // y coordinate
   const int  flags       // flags
   )
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Return Value

true - event has been processed, otherwise - false.

Name

Gets control name.

```
string  Name()  const
```

Return Value

Control name.

ControlsTotal

Gets the number of controls in the container.

```
int  ControlsTotal()  const
```

Return Value

Number of controls in container.

Note

The base class method does not have the container, it provides the access to container for its heirs and always returns 0.

Control

Gets the control from the container by index.

```
CWnd*  Control(
   const int  ind      // index
   )  const
```

Parameters

ind

[in] Control index.

Return Value

A pointer to the control.

Note

The base class method does not have the container, it provides the access to container for its heirs and always returns NULL.

ControlFind

Gets the control from container by specified ID.

```
virtual CWnd*  ControlFind(
   const long  id      // ID
   )
```

Parameters

id

[in] Identifier of the control to find.

Return Value

Pointer to the control from the container.

Note

The base class method does not have the container, it provides the access to container for its heirs. If the specified ID matches the container's one, it returns a pointer to itself (this).

Rect

Gets the pointer to the CRect class object.

```
const CRect*  Rect()  const
```

Return Value

Pointer to the CRect class object.

Left (Get Method)

Gets the X coordinate of the upper-left corner of the control.

```
int  Left()
```

Return Value

X coordinate of the upper-left corner of the control.

Left (Set Method)

Sets the X coordinate of the upper-left corner of the control.

```
void  Left(
   const int  x       // coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Return Value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner of the control.

```
int  Top()
```

Return Value

Y coordinate of the upper-left corner of the control.

Top (Set Method)

Sets the Y coordinate of the upper-left corner of the control.

```
void  Top(
   const int  y       // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Return Value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner of the control.

```
int  Right()
```

Return Value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the X coordinate of the lower-right corner of the control.

```
void  Right(
   const int  x       // x coordinate
   )
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Return Value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner of the control.

```
int  Bottom()
```

Return Value

Y coordinate of the lower-right corner of the control.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner of the control.

```
void  Bottom(
   const int  y       // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Return Value

None.

Width (Get Method)

Gets the control width.

```
int  Width()
```

Return Value

Width of the control.

Width (Set Method)

Sets the width of the control.

```
virtual bool  Width(
   const int  w       // width
   )
```

Parameters

w

[in] New width.

Return Value

true - successful, otherwise - false.

Height (Get Method)

Gets the control height.

```
int  Height()
```

Return Value

Height of the control.

Height (Set Method)

Sets new height of the control.

```
virtual bool  Height(
   const int  h       // height
   )
```

Parameters

h

[in] New height.

Return Value

true - successful, otherwise - false.

Move

Performs absolute displacement of a control.

```
void  Move(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left point.

y

[in] New Y coordinate of the upper-left point.

Return Value

true - successful, otherwise - false.

Shift

Performs the relative shift of a control.

```
void  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

true - successful, otherwise - false.

Resize

Sets a size of the control.

```
virtual bool  Resize(
   const int  w,
   const int  h
   )
```

Parameters

w

[in] New width.

h

[in] New height.

Return Value

true - successful, otherwise - false.

Contains

Checks if the point is inside the control area of the chart.

```
bool  Contains(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Return Value

true - the point is inside the area (including borders), otherwise - false.

Contains

Checks if the specified control is inside the control area of the chart.

```
bool  Contains(
   const CWnd*  control      // pointer
   )  const
```

Parameters

control

[in] Object pointer.

Return Value

true - the specified control is inside the area (including borders), otherwise - false.

Alignment

Sets alignment parameters of the control.

```
void  Alignment(
   const int  flags,       // flags
   const int  left,        // offset
   const int  top,         // offset
   const int  right,       // offset
   const int  bottom       // offset
   )
```

Parameters

flags

[in] Alignment flags.

left

[in] Fixed offset from the left border.

top

[in] Fixed offset from the top border.

right

[in] Fixed offset from the right border.

bottom

[in] Fixed offset from the bottom border.

Return Value

None.

Note

Alignement flags:

```
enum WND_ALIGN_FLAGS
{
   WND_ALIGN_NONE=0,                                   // no align
   WND_ALIGN_LEFT=1,                                   // align left
   WND_ALIGN_TOP=2,                                    // align top
   WND_ALIGN_RIGHT=4,                                  // align right
   WND_ALIGN_BOTTOM=8,                                 // align bottom
   WND_ALIGN_WIDTH = WND_ALIGN_LEFT|WND_ALIGN_RIGHT,   // align width
   WND_ALIGN_HEIGHT=WND_ALIGN_TOP|WND_ALIGN_BOTTOM,    // align height
   WND_ALIGN_CLIENT=WND_ALIGN_WIDTH|WND_ALIGN_HEIGHT,  // align height and width
   }
```

Align

Performs control alignment in the specified chart area.

```
virtual bool  Align(
   const CRect*   rect       // pointer
   )
```

Parameters

rect

[in] Pointer to the object with chart area coordinates.

Return Value

true - successful, otherwise - false.

Note

The alignment parameters must be specified (no alignment by default).

Id (Get Method)

Gets the control ID.

```
long  Id()  const
```

Return Value

The control identifier.

Id (Set Method)

Sets the value of the control ID.

```
virtual long  Id(
   const long  id      // identifier
   )
```

Parameters

id

[in] New value of the control identifier.

Return Value

None.

IsEnabled

Checks the control ability to respond to user's actions.

```
bool  IsEnabled()  const
```

Return Value

true - control is enabled, otherwise - false.

Enable

Enables the control ability to respond to user's actions.

```
virtual bool  Enable()
```

Return Value

true - successful, otherwise - false.

Note

If the control is enabled, it is able to process the external events.

Disable

Disables the control ability to respond to user's actions.

```
virtual bool  Disable()
```

Return Value

true - successful, otherwise - false.

Note

The disabled control is not able to process the external events.

IsVisible

Checks whether the control is visible.

```
bool  IsVisible()  const
```

Return Value

true - control is shown on the chart, otherwise - false.

Visible

Sets the visibility flag.

```
virtual bool  Visible(
   const bool  flag       // flag
   )
```

Parameters

flag

[in] New flag.

Return Value

true - successful, otherwise - false.

Show

Shows the control.

```
virtual bool  Show()
```

Return Value

true - successful, otherwise - false.

Hide

Hides the control.

```
virtual bool  Hide()
```

Return Value

true - successful, otherwise - false.

IsActive

Gets a value indicating whether the control is active.

```
bool  IsActive()  const
```

Return Value

true - control is active, otherwise - false.

Activate

Activates the control.

```
virtual bool  Activate()
```

Return Value

true - successful, otherwise - false.

Note

The control becomes active when the mouse cursor is hovering over it.

Deactivate

Deactivates the control.

```
virtual bool  Deactivate()
```

Return Value

true - successful, otherwise - false.

Note

The control becomes inactive when the mouse cursor is out off the control.

StateFlags (Get Method)

Gets the control state flags.

```
int  StateFlags()
```

Return Value

The control state flags.

StateFlags (Set Method)

Sets the control state flags.

```
virtual void  StateFlags(
   const int  flags      // flags
   )
```

Parameters

flags

[in] New control state flags.

Return Value

None.

StateFlagsSet

Sets the control state flags.

```
virtual void  StateFlagsSet(
   const int  flags       // flags
   )
```

Parameters

flags

[in] Flags to set.

Return Value

None.

StateFlagsReset

Resets the control state flags.

```
virtual void  StateFlagsReset(
   const int  flags       // flags
   )
```

Parameters

flags

[in] Flags to reset.

Return Value

None.

PropFlags (Get Method)

Gets the control properties flags.

```
void  PropFlags(
   const int  flags      // flags
   )
```

Return Value

The control properties flags.

PropFlags (Set Method)

Sets the control properties flags.

```
virtual void  PropFlags(
   const int  flags      // flags
   )
```

Parameters

flags

[in] New flags.

Return Value

None.

PropFlagsSet

Sets the control properties flags.

```
virtual void  PropFlagsSet(
   const int  flags      // flags
   )
```

Parameters

flags

[in] Flags to set.

Return Value

None.

PropFlagsReset

Resets the control properties flags.

```
virtual void  PropFlagsReset(
   const int  flags       // flags
   )
```

Parameters

flags

[in] Flags to reset.

Return Value

None.

MouseX (Set Method)

Saves the mouse X coordinate.

```
void  MouseX(
   const int   value       // coordinate
   )
```

Parameters

value

[in] The X coordinate of the mouse.

Return Value

None.

MouseX (Get Method)

Gets the saved X coordinate of the mouse.

```
int  MouseX()
```

Return Value

The X coordinate of the mouse.

MouseY (Set Method)

Saves the mouse Y coordinate.

```
void  MouseY(
   const int  value      // coordinate
   )
```

Parameters

value

[in] The Y coordinate of the mouse.

Return Value

None.

MouseY (Get Method)

Gets the saved Y coordinate of the mouse.

```
int  MouseY()
```

Return Value

The Y coordinate of the mouse.

MouseFlags (Set Method)

Saves the state of mouse buttons.

```
virtual void  MouseFlags(
   const int  value      // state
   )
```

Parameters

value

[in] State of mouse buttons.

Return Value

None.

MouseFlags (Get Method)

Gets the saved state of mouse buttons.

```
int  MouseFlags()
```

Return Value

State of mouse buttons.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates the control.

```
bool  MouseFocusKill(
   const long   id=CONTROLS_INVALID_ID      // id
   )
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Return Value

The control deactivation result.

OnCreate

The virtual handler of the internal "Create" event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDestroy

The virtual handler of the internal "Destroy" event.

```
virtual bool  OnDestroy()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnMove

The virtual handler of the internal "Move" event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnResize

The virtual handler of the internal "Resize" event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnEnable

The virtual handler of the internal "Enable" event (if enabled, it can respond to user interaction).

```
virtual bool  OnEnable()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDisable

The virtual handler of the internal "Disable" event (if disabled, it cannot respond to user interaction).

```
virtual bool  OnDisable()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnShow

The virtual handler of the internal "Show" event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHide

The virtual handler of the internal "Hide" event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnActivate

The virtual handler of the internal "Activate" event.

```
virtual bool  OnActivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The virtual handler of the internal "Deactivate" event.

```
virtual bool  OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnClick

The virtual handler of the internal "Click" event (mouse click).

```
virtual bool  OnClick()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of the internal "Change" event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnMouseDown

The virtual handler of the "MouseDown" (mouse click) event.

```
virtual bool  OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The virtual handler of the "MouseUp" (mouse button release) event.

```
virtual bool  OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

OnDragStart

The virtual handler of the "DragStart" (start dragging) event.

```
virtual bool  OnDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "DragStart" event occurs at the start of control dragging operation.

OnDragProcess

The virtual handler of the "DragProcess" (dragging) event.

```
virtual bool  OnDragProcess(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "DragProcess" event occurs when the control is dragged.

OnDragEnd

The virtual handler of the "DragEnd" event.

```
virtual bool  OnDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "DragEnd" event occurs when control drag process is finished.

DragObjectCreate

Creates drag object.

```
virtual bool  DragObjectCreate()
```

Return Value

true - successful, otherwise - false.

Note

true - event processed, otherwise - false.

DragObjectDestroy

Destroys drag object.

```
virtual bool  DragObjectDestroy()
```

Return Value

true - successful, otherwise - false.

CWndObj

CWndObj is a base class for simple controls (based on chart objects) of the Standard Library.

Description

CWndObj class implements base methods of the simple control.

Declaration

```
class CWndObj : public CWnd
```

Title

```
#include <Controls\WndObj.mqh>
```

Inheritance hierarchy

CObject
  CWnd
    CWndObj

Direct descendants

CBmpButton, CButton, CEdit, CLabel, CPanel, CPicture

Class Methods by Groups

Chart events processing	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/sets the OBJPROP_TEXT property of the chart object
Color	Gets/sets the OBJPROP_COLOR property of the chart object
ColorBackground	Gets/sets the OBJPROP_BGCOLOR property of the chart object
ColorBorder	Gets/sets the OBJPROP_BORDER_COLOR property of the chart object
Font	Gets/sets the OBJPROP_FONT property of the chart object
FontSize	Gets/sets the OBJPROP_FONTSIZE property of the chart object
ZOrder	Gets/sets the OBJPROP_ZORDER property of the chart object
Chart objects event handlers	
OnObjectCreate	CHARTEVENT_OBJECT_CREATE event handler
OnObjectChange	CHARTEVENT_OBJECT_CHANGE event handler
OnObjectDelete	CHARTEVENT_OBJECT_DELETE event handler

Chart events processing	
OnObjectDrag	CHARTEVENT_OBJECT_DRAG event handler
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnDestroy	"Destroy" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Create, Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets the OBJPROP_TEXT (text) property of the chart object.

```
string  Text()
```

Return Value

The value of the OBJPROP_TEXT property.

Text (Set method)

Sets the OBJPROP_TEXT (text) property of the chart object.

```
bool  Text(
   const string  value       // value
   )
```

Parameters

value

[in] New value of the OBJPROP_TEXT property.

Return Value

true - successful, otherwise - false.

Color (Get method)

Gets the OBJPROP_COLOR (color) property of the chart object.

```
color  Color()
```

Return Value

The value of the OBJPROP_COLOR property.

Color (Set method)

Sets the OBJPROP_COLOR (color) propery of the chart object.

```
bool  Color(
   const color  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_COLOR property.

Return Value

true - successful, otherwise - false.

ColorBackground (Get method)

Gets the OBJPROP_BGCOLOR (background color) of the chart object.

```
color  ColorBackground()
```

Return Value

The value of the OBJPROP_BGCOLOR property.

ColorBackground (Set method)

Sets the OBJPROP_BGCOLOR (background color) property of the chart object.

```
bool  ColorBackground(
   const color  value       // value
   )
```

Parameters

value

[in] New value of the OBJPROP_BGCOLOR property.

Return Value

true - successful, otherwise - false.

ColorBorder (Get method)

Gets the OBJPROP_BORDER_COLOR (border color) property of the chart object.

```
color  ColorBorder()
```

Return Value

The value of the OBJPROP_BORDER_COLOR property.

ColorBorder (Set method)

Sets the OBJPROP_BORDER_COLOR (border color) property of the chart object.

```
bool  ColorBorder(
   const color  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_BORDER_COLOR property.

Return Value

true - successful, otherwise - false.

Font (Get method)

Gets the OBJPROP_FONT (font) property of the chart object.

```
string  Font()
```

Return Value

The value of the OBJPROP_FONT property.

Font (Set method)

Sets the OBJPROP_FONT (font) property of the chart object.

```
bool  Font(
   const string  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_FONT property.

Return Value

true - successful, otherwise - false.

FontSize (Get method)

Gets the OBJPROP_FONTSIZE (font size) property of the chart object.

```
int  FontSize()
```

Return Value

The value of the OBJPROP_FONTSIZE property.

FontSize (Set method)

Sets the OBJPROP_FONTSIZE (font size) property of the chart object.

```
bool  FontSize(
   const int  value       // value
   )
```

Parameters

value

[in] New value of the OBJPROP_FONTSIZE property.

Return Value

true - successful, otherwise - false.

ZOrder (Get method)

Gets the OBJPROP_ZORDER property of the chart object.

```
long  ZOrder()
```

Return Value

The value of the OBJPROP_ZORDER property.

ZOrder (Set method)

Sets the OBJPROP_ZORDER property of the chart object.

```
bool  ZOrder(
   const long  value       // value
   )
```

Parameters

value

[in] New value of the OBJPROP_ZORDER property.

Return Value

true - successful, otherwise - false.

OnObjectCreate

The virtual handler of chart object CHARTEVENT_OBJECT_CREATE event.

```
virtual bool  OnObjectCreate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnObjectChange

The virtual handler of chart object CHARTEVENT_OBJECT_CHANGE event.

```
virtual bool  OnObjectChange()
```

Return Value

true - event processed, otherwise - false.

OnObjectDelete

The virtual handler of chart object CHARTEVENT_OBJECT_DELETE event.

```
virtual bool  OnObjectDelete()
```

Return Value

true - event processed, otherwise - false.

OnObjectDrag

The virtual handler of chart object CHARTEVENT_OBJECT_DRAG event.

```
virtual bool  OnObjectDrag()
```

Return Value

true - event processed, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the OBJPROP_TEXT property) event.

```
virtual bool  OnSetText()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetColor

The virtual handler of control "SetColor" (change of the OBJPROP_COLOR property) event.

```
virtual bool  OnSetColor()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetFont

The virtual handler of "SetFont" (change of the OBJPROP_FONT property) event.

```
virtual bool  OnSetFont()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetFontSize

The virtual handler of "SetFontSize" (change of the OBJPROP_FONTSIZE property) event.

```
virtual bool  OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnSetZOrder

The virtual handler of "SetZOrder" (change of the OBJPROP_ZORDER property) event.

```
virtual bool  OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDestroy

The virtual handler of control "Destroy" internal event.

```
virtual bool  OnDestroy()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

CWndContainer

CWndContainer is a base class for a complex control (containing dependent controls) of the Standard library.

Description

CWndContainer class implements base methods of the complex control.

Declaration

```
class CWndContainer : public CWnd
```

Title

```
#include <Controls\WndContainer.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer

Direct descendants

CCheckBox, CComboBox, CDateDropList, CDatePicker, CDialog, CRadioButton, CScroll, CSpinEdit, CWndClient

Class Methods by Groups

Destroy	
Destroy	Destroys all the container controls
Chart event handlers	
OnEvent	Event handler of all chart events
OnMouseEvent	The CHARTEVENT_MOUSE_MOVE event handler
Access to container	
ControlsTotal	Gets the number of controls in the container
Control	Gets control by index
ControlFind	Gets control by ID
Add/Delete	
Add	Adds control to a group (container)
Delete	Deletes control from a group (container)
Geometry	

Destroy	
Move	Performs a absolute displacement of an element group
Shift	Performs a relative displacement of an element group
Identification	
Id	Sets the ID for all controls of the container
State	
Enable	Enables all controls of the container
Disable	Disables all controls of the container
Show	Shows all controls of the container
Hide	Hides all controls of the container
Mouse operations	
MouseFocusKill	Kills mouse focus
File operations	
Save	Saves container information to file
Load	Loads container information from file
Internal event handlers	
OnResize	"Resize" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Create, Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Destroy

Destroys all the container controls.

```
virtual bool  Destroy()
```

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Return Value

true - event processed, otherwise - false.

OnMouseEvent

Mouse event handler.

```
virtual bool  OnMouseEvent(
   const int  x,          // x coordinate
   const int  y,          // y coordinate
   const int  flags       // flags
   )
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Return Value

true - event processed, otherwise - false.

ControlsTotal

Gets the number of controls in the container.

```
int  ControlsTotal()  const
```

Return Value

Number of controls in the contrainer.

Control

Gets control from the container by index.

```
CWnd*  Control(
   const int  ind       // index
   )  const
```

Parameters

ind

[in] Index of the control needed.

Return Value

Pointer to the control, otherwise NULL if the control is not found.

ControlFind

Gets control from the container by identifier.

```
virtual CWnd*  ControlFind(
   const long  id       // id
   )
```

Parameters

id

[in] Control ID.

Return Value

Pointer to the control, otherwise NULL if the control is not found.

Add

Adds a control to a group (container).

```
bool  Add(
   CWnd&  control        // reference
   )
```

Parameters

control

[in] Control to add, passed by reference.

Return Value

true - successful, otherwise - false.

Delete

Deletes control from a group (container).

```
bool  Delete(
   CWnd&   control       // reference
   )
```

Parameters

control

[in] Control to delete, passed by reference.

Return Value

true - successful, otherwise - false.

Move

Performs an absolute displacement of all controls of the container.

```
virtual bool  Move(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

y

[in] New Y coordinate of the upper-left corner.

Return Value

true - successful, otherwise - false.

Shift

Performs the relative shift of the coordinates for all controls of the container.

```
virtual bool  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Return Value

true - successful, otherwise - false.

Id

Sets the ID for all controls of the container.

```
virtual long  Id(
   const long  id      // identifier
   )
```

Parameters

id

[in] Base group identifier.

Return Value

Number of identifiers used by container controls.

Enable

Enables all the controls of the container.

```
virtual bool  Enable()
```

Return Value

true - successful, otherwise - false.

Disable

Disables all controls of the container.

```
virtual bool  Disable()
```

Return Value

true - successful, otherwise - false.

Show

Shows all controls of the container.

```
virtual bool  Show()
```

Return Value

true - successful, otherwise - false.

Hide

Hides all controls of the container.

```
virtual bool  Hide()
```

Return Value

true - successful, otherwise - false.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates all controls in the container.

```
bool  MouseFocusKill(
   const long  id=CONTROLS_INVALID_ID      // id
   )
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Return Value

The controls deactivation result.

Save

Saves container information to file.

```
virtual bool  Save(
   const int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened for writing.

Return Value

true - successful, otherwise - false.

Load

Loads container information from file

```
virtual bool  Load(
   const int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of the binary file previously opened for reading.

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnActivate

The virtual handler of control "Activate" internal event.

```
virtual bool  OnActivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The virtual handler of control "Deactivate" internal event.

```
virtual bool  OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

CLabel

CLabel is class of the simple control based on "Text label" chart object.

Description

CLabel is intended for creation of simple non-editable text labels.

Declaration

```
class CLabel : public CWndObj
```

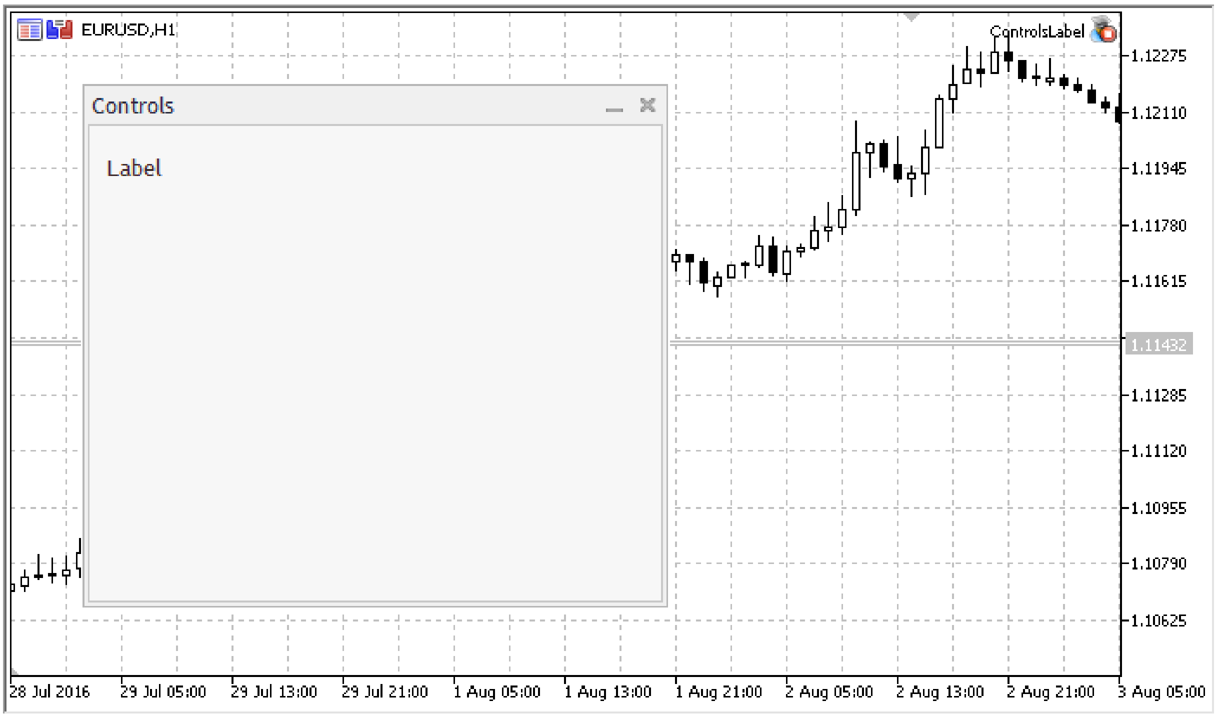
Title

```
#include <Controls\Label.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CLabel

Result of the code provided below:


EURUSD,H1
ControlsLabel
Controls
Label
1.12275
1.12110
1.11945
1.11780
1.11615
1.11432
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with text label:

```
//+------------------------------------------------------------------+
//|                                                 ControlsLabel.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CLabel"
#include <Controls\Dialog.mqh>
#include <Controls\Label.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CLabel            m_label;                         // CLabel object
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
protected:
   //--- create dependent controls
   bool              CreateLabel(void);
   //--- handlers of the dependent controls events
   void              OnClickLabel(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)

EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
```

```
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateLabel())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CLabel"                                              |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateLabel(void)
  {
//--- coordinates
   int x1=INDENT_RIGHT;
   int y1=INDENT_TOP+CONTROLS_GAP_Y;
   int x2=x1+100;
   int y2=y1+20;
//--- create
   if(!m_label.Create(m_chart_id,m_name+"Label",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_label.Text("Label"))
      return(false);
   if(!Add(m_label))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
```

```
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CLabel control.

```
virtual bool  Create(
   const long    chart,       // chart ID
   const string  name,        // name
   const int     subwin,      // chart subwindow
   const int     x1,          // x1 coordinate
   const int     y1,          // y1 coordinate
   const int     x2,          // x2 coordinate
   const int     y2           // y2 coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the OBJPROP_TEXT property) event.

```
virtual bool  OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the OBJPROP_COLOR property) event.

```
virtual bool  OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the OBJPROP_FONT property) event.

```
virtual bool  OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event.

```
virtual bool  OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

CBmpButton

CBmpButton is class of the simple control based on "Bitmap label" chart object.

Description

CBmpButton is intended for creation of buttons with graphic image.

Declaration

```
class CBmpButton : public CWndObj
```

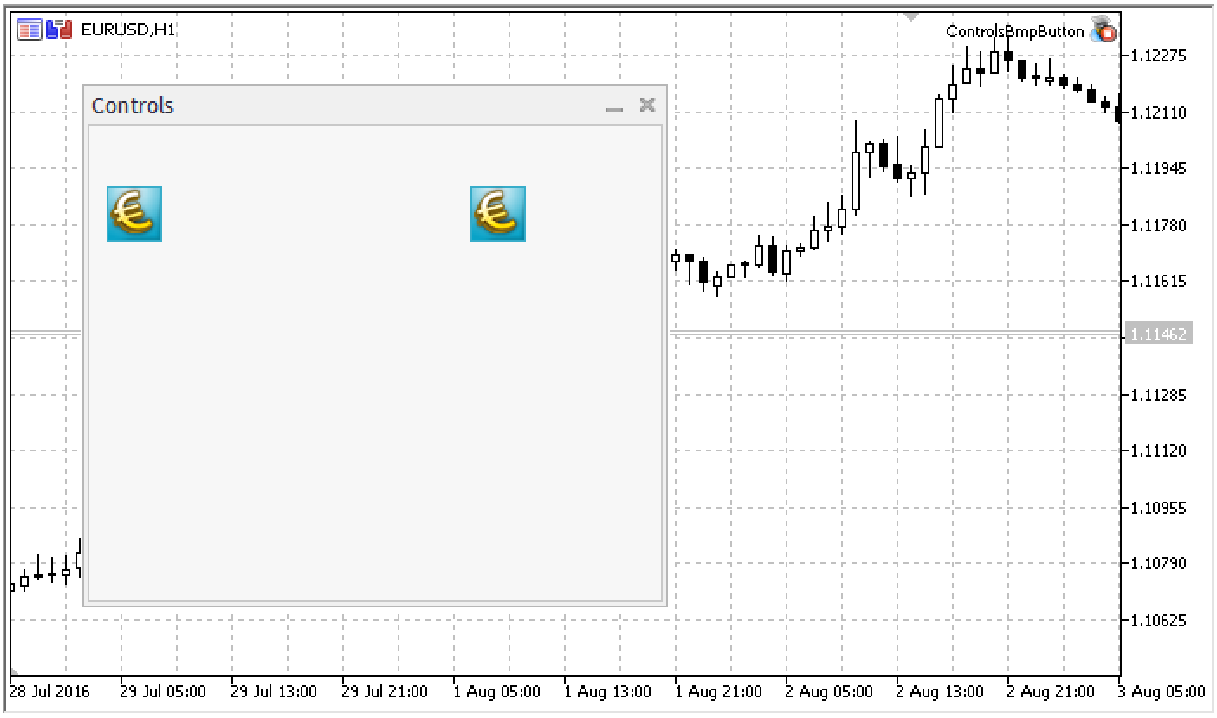
Title

```
#include <Controls\BmpButton.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CBmpButton

Result of the code provided below:


EURUSD,H1
ControlsBmpButton
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11462
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Properties	
Border	Gets/sets the "Border" property of the control
BmpNames	Sets the name of bmp files of the control
BmpOffName	Gets/sets the name of bmp file for the OFF state
BmpOnName	Gets/sets the name of bmp file for the ON state
BmpPassiveName	Gets/sets the name of bmp file for the passive state
BmpActiveName	Gets/sets the name of bmp file for the active state
State	
Pressed	Gets/sets the state of the control
Locking	Gets/sets the "Locking" property of the control
Internal event handlers	
OnSetZOrder	"SetZOrder" internal event handler
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnChange	"Change" internal event handler
OnActivate	"Activate" internal event handler
OnDeactivate	"Deactivate" internal event handler
OnMouseDown	"MouseDown" internal event handler
OnMouseUp	"MouseUp" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with Bitmap label:

```
//+------------------------------------------------------------------+
//|                                            ControlsBmpButton.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CBmpButton"
#include <Controls\Dialog.mqh>
#include <Controls\BmpButton.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CBmpButton        m_bmpbutton1;                    // CBmpButton object
   CBmpButton        m_bmpbutton2;                    // CBmpButton object
```

```
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateBmpButton1(void);
   bool              CreateBmpButton2(void);
   //--- handlers of the dependent controls events
   void              OnClickBmpButton1(void);
   void              OnClickBmpButton2(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_bmpbutton1,OnClickBmpButton1)
ON_EVENT(ON_CLICK,m_bmpbutton2,OnClickBmpButton2)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateBmpButton1())
      return(false);
   if(!CreateBmpButton2())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "BmpButton1" button                                   |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateBmpButton1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_bmpbutton1.Create(m_chart_id,m_name+"BmpButton1",m_subwin,x1,y1,x2,y2))
      return(false);
//--- sets the name of bmp files of the control CBmpButton
   m_bmpbutton1.BmpNames("\\Images\\euro.bmp","\\Images\\dollar.bmp");
   if(!Add(m_bmpbutton1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "BmpButton2" fixed button                             |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateBmpButton2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_bmpbutton2.Create(m_chart_id,m_name+"BmpButton2",m_subwin,x1,y1,x2,y2))
      return(false);
//--- sets the name of bmp files of the control CBmpButton
   m_bmpbutton2.BmpNames("\\Images\\euro.bmp","\\Images\\dollar.bmp");
   if(!Add(m_bmpbutton2))
      return(false);
   m_bmpbutton2.Locking(true);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickBmpButton1(void)
  {
   Comment(__FUNCTION__);
  }
```

```
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickBmpButton2(void)
  {
   if(m_bmpbutton2.Pressed())
      Comment(__FUNCTION__+" State of the control is: On");
   else
      Comment(__FUNCTION__+" State of the control is: Off");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID  
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CBmpButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int  Border()  const
```

Return Value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool  Border(
   const int  value       // value
   )
```

Parameters

value

[in] New value of the "Border" property.

Return Value

true - successful, otherwise - false.

BmpNames

Sets the name of bmp files of the control

```
bool  BmpNames(
   const string  off="",      // file name
   const string  on=""        // file name
   )
```

Parameters

off=""

[in] Name of bmp file for OFF state.

on=""

[in] Name of bmp file for ON state.

Return Value

true - successful, otherwise - false.

BmpOffName (Get method)

Gets the name of bmp file for OFF state.

```
string  BmpOffName()  const
```

Return Value

Name of bmp file for OFF state.

BmpOffName (Set method)

Sets the name of bmp file for OFF state.

```
bool  BmpOffName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for OFF state.

Return Value

true - successful, otherwise - false.

BmpOnName (Get method)

Gets the name of bmp file for ON state.

```
string  BmpOnName()  const
```

Return Value

Name of bmp file for ON state.

BmpOnName (Set method)

Sets the name of bmp file for ON state.

```
bool  BmpOnName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for ON state.

Return Value

true - successful, otherwise - false.

BmpPassiveName (Get method)

Gets the name of bmp file for the control passive state.

```
string  BmpPassiveName()  const
```

Return Value

Name of bmp file for the control passive state.

BmpPassiveName (Set method)

Sets the name of bmp file for the passive state.

```
bool  BmpPassiveName(
   const string  name       // file name
   )
```

Parameters

name

[in] Name of bmp file for the control passive state.

Return Value

true - successful, otherwise - false.

BmpActiveName (Get method)

Gets the name of bmp file for the active state.

```
string  BmpActiveName()  const
```

Return Value

Name of bmp file for the active state.

Note

The control becomes active when the mouse cursor is hovering over it.

BmpActiveName (Set method)

Sets the name of bmp file for the active state.

```
bool  BmpActiveName(
   const string  name       // file name
   )
```

Parameters

name

[in] Name of bmp file for the active state.

Return Value

true - successful, otherwise - false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool  Pressed()  const
```

Return Value

Control state.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool  Pressed(
   const bool   pressed       // state
   )
```

Parameters

pressed

[in] New control state.

Return Value

true - successful, otherwise - false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool  Locking()  const
```

Return Value

The value of "Locking" property.

Locking (Set method)

Sets new value of the "Locking" property of the control.

```
void  Locking(
   const bool  locking      // value
   )
```

Parameters

locking

[in] New value of "Locking" property.

Return Value

None.

OnSetZOrder

The virtual handler of control "SetZOrder" (change of the OBJPROP_ZORDER property) event.

```
virtual bool  OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

OnActivate

The virtual handler of control "Activate" internal event.

```
virtual bool  OnActivate()
```

Return Value

true - event processed, otherwise - false.

OnDeactivate

The virtual handler of control "Deactivate" internal event.

```
virtual bool  OnDeactivate()
```

Return Value

true - event processed, otherwise - false.

OnMouseDown

The virtual handler of control "MouseDown" event.

```
virtual bool  OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

OnMouseUp

The virtual handler of control "MouseUp" (mouse button release) event.

```
virtual bool  OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

CButton

CButton is a class of a simple control based on "Button" chart object.

Description

CButton class is intended for creation of simple buttons.

Declaration

```
   class CButton : public CWndObj
```

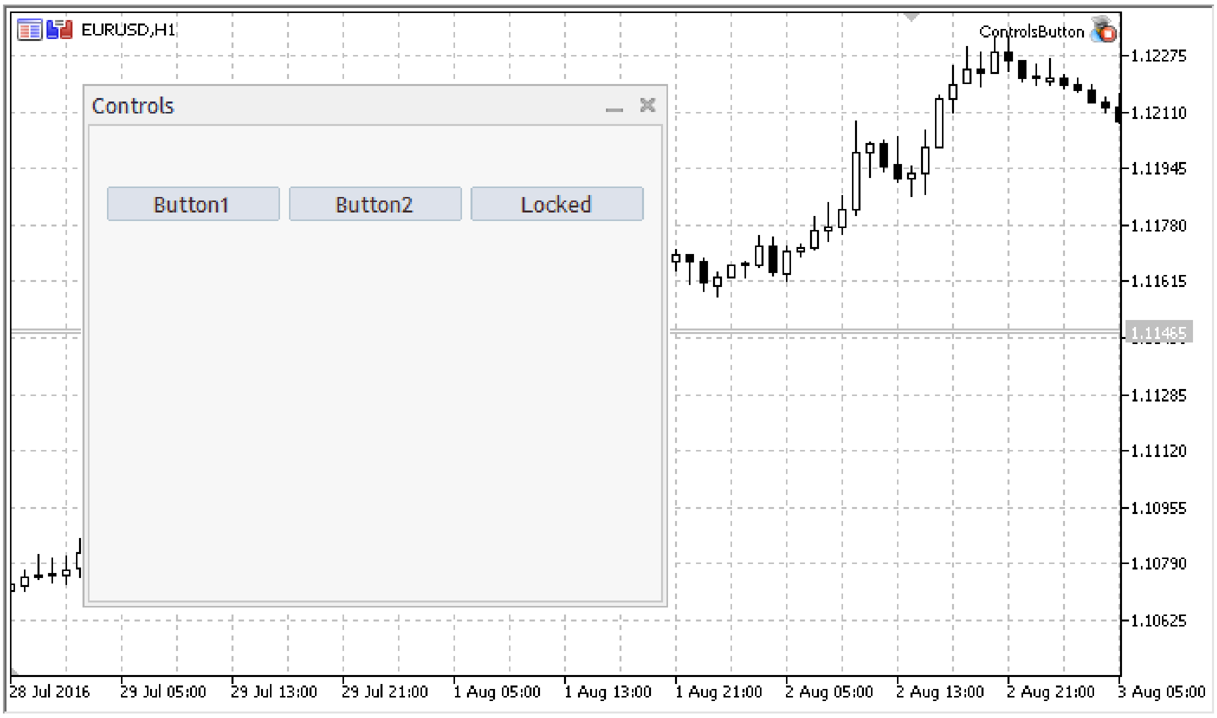
Title

```
   #include <Controls\Button.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CButton

Result of the code provided below:


EURUSD,H1
ControlsButton
Controls
Button1
Button2
Locked
1.12275
1.12110
1.11945
1.11780
1.11615
1.11465
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
State	
Pressed	Gets/sets the "Pressed" property
Locking	Gets/sets the "Locking" property
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnMouseDown	"MouseDown" internal event handler
OnOnMouseUp	"MouseUp" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with button:

```
//+------------------------------------------------------------------+
//|                                               ControlsButton.mq5 |
//|                          Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CButton"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CButton           m_button1;                       // the button object
   CButton           m_button2;                       // the button object
   CButton           m_button3;                       // the fixed button object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
protected:
   //--- create dependent controls
   bool              CreateButton1(void);
   bool              CreateButton2(void);
   bool              CreateButton3(void);
   //--- handlers of the dependent controls events
   void              OnClickButton1(void);
   void              OnClickButton2(void);
   void              OnClickButton3(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_button1,OnClickButton1)
ON_EVENT(ON_CLICK,m_button2,OnClickButton2)
ON_EVENT(ON_CLICK,m_button3,OnClickButton3)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateButton1())
      return(false);
   if(!CreateButton2())
      return(false);
   if(!CreateButton3())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button1" button                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButton1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_button1.Create(m_chart_id,m_name+"Button1",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_button1.Text("Button1"))
      return(false);
   if(!Add(m_button1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button2" button                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButton2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_button2.Create(m_chart_id,m_name+"Button2",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_button2.Text("Button2"))
      return(false);
   if(!Add(m_button2))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Button3" fixed button                                |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateButton3(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+BUTTON_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
```

```
   if(!m_button3.Create(m_chart_id,m_name+"Button3",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_button3.Text("Locked"))
      return(false);
   if(!Add(m_button3))
      return(false);
   m_button3.Locking(true);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton1(void)
  {
   Comment(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton2(void)
  {
   Comment(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickButton3(void)
  {
   if(m_button3.Pressed())
      Comment(__FUNCTION__+" State of the control: On");
   else
      Comment(__FUNCTION__+" State of the control: Off");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
```

```
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID  
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool  Pressed()  const
```

Return Value

"Pressed" property of the control.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool  Pressed(
   const bool  pressed      // state
   )
```

Parameters

pressed

[in] New control state.

Return Value

true - successful, otherwise - false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool  Locking()  const
```

Return Value

The value of "Locking" property.

Locking (Set method)

Sets the value of "Locking" property of the control.

```
void  Locking(
   const bool   locking       // value
   )
```

Parameters

locking

[in] New value of "Locking" property of the control.

Return Value

None.

OnSetText

The virtual handler of control "SetText" (change of the OBJPROP_TEXT property) event.

```
virtual bool  OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the OBJPROP_COLOR property) event.

```
virtual bool  OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the OBJPROP_FONT property) event.

```
virtual bool  OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event.

```
virtual bool  OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnMouseDown

The virtual handler of control "MouseDown" event handler.

```
virtual bool  OnMouseDown()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The virtual handler of control "MouseUp" (left mouse button release) event.

```
virtual bool  OnMouseUp()
```

Return Value

true - event processed, otherwise - false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

CEdit

CEdit is class of the simple control based on "Edit" chart object.

Description

CEdit class in intended for creation of controls, in which the user can enter text.

Declaration

```
class CEdit : public CWndObj
```

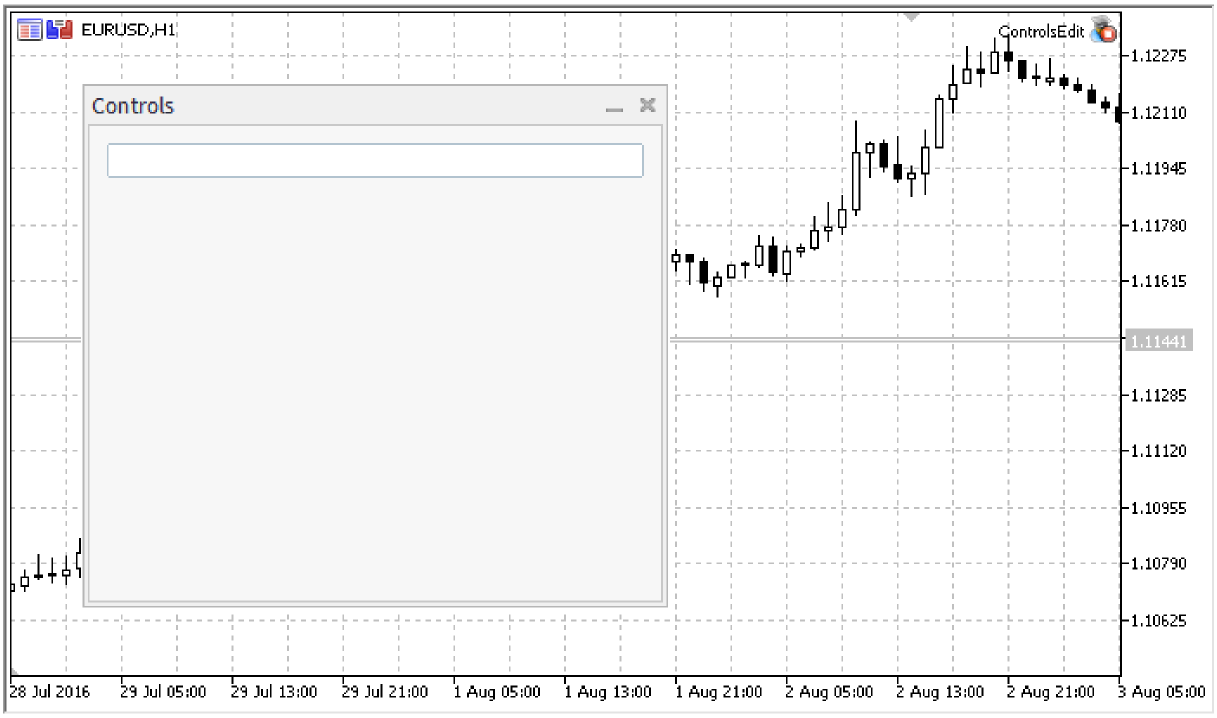
Title

```
#include <Controls\Edit.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CEdit

Result of the code provided below:


EURUSD,H1
ControlsEdit
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11441
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Properties	
ReadOnly	Gets/sets the "ReadOnly" property
TextAlign	Gets/sets the "TextAlign" property
Chart object event handlers	
OnObjectEndEdit	The CHARTEVENT_OBJECT_ENDEDIT event handler (virtual)
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnChange	"Change" internal event handler
OnClick	"Click" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with Edit control:

```
//+------------------------------------------------------------------+
//|                                                 ControlsEdit.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CEdit"
#include <Controls\Dialog.mqh>
#include <Controls\Edit.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CEdit             m_edit;                          // CEdit object

public:
```

```
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler

protected:
   //--- create dependent controls
   bool              CreateEdit(void);
  };
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateEdit())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the display field                                         |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateEdit(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=ClientAreaWidth()-INDENT_RIGHT;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_edit.Create(m_chart_id,m_name+"Edit",m_subwin,x1,y1,x2,y2))
      return(false);
//--- allow editing the content
```

```
   if(!m_edit.ReadOnly(false))
      return(false);
   if(!Add(m_edit))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates CEdit control.

```
virtual bool  Create(
   const long     chart,        // chart ID
   const string   name,         // name
   const int      subwin,       // chart subwindow
   const int      x1,           // coordinate
   const int      y1,           // coordinate
   const int      x2,           // coordinate
   const int      y2            // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

ReadOnly (Get Method)

Gets the "ReadOnly" property of the control.

```
bool  ReadOnly()
```

Return Value

The value of "ReadOnly" property.

ReadOnly (Set Method)

Sets the value of "ReadOnly" property of the control.

```
bool  ReadOnly(
   const bool  flag      // value
   )
```

Parameters

flag

[in] New value of "ReadOnly" property.

Return Value

true - successful, otherwise - false.

TextAlign (Get method)

Gets the value of "TextAlign" property (text alignment mode) of the control.

```
ENUM_ALIGN_MODE  TextAlign() const
```

Return Value

Value of "TextAlign" property of the control.

TextAlign (Set method)

Sets new value of "TextAlign" property (text aligment mode) of the control.

```
bool  TextAlign(
   ENUM_ALIGN_MODE  align      // property value
   )
```

Parameters

align

[in] New value of "TextAlign" property.

Return Value

true - successful, false - cannot change the property.

OnObjectEndEdit

The virtual handler of CHARTEVENT_OBJECT_ENDEDIT chart object event.

```
virtual bool  OnObjectEndEdit()
```

Return Value

true - event processed, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the OBJPROP_TEXT property) event.

```
virtual bool  OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColor

The virtual handler of control "SetColor" (change of the OBJPROP_COLOR property) event.

```
virtual bool  OnSetColor()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetFont

The virtual handler of control "SetFont" (change of the OBJPROP_FONT property) event.

```
virtual bool  OnSetFont()
```

Return Value

true - event processed, otherwise - false.

OnSetFontSize

The virtual handler of control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event.

```
virtual bool  OnSetFontSize()
```

Return Value

true - event processed, otherwise - false.

OnSetZOrder

The virtual handler of control "SetZOrder" (change of the OBJPROP_ZORDER property) event.

```
virtual bool  OnSetZOrder()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

OnClick

The virtual handler of control "Click" (mouse button click) internal event.

```
virtual bool  OnClick()
```

Return Value

true - event processed, otherwise - false.

CPanel

CPanel is a class of the simple control based on "Rectangle label" chart object.

Description

CPanel class is intended to combine the controls with similar functions in the group.

Declaration

```
   class CPanel : public CWndObj
```

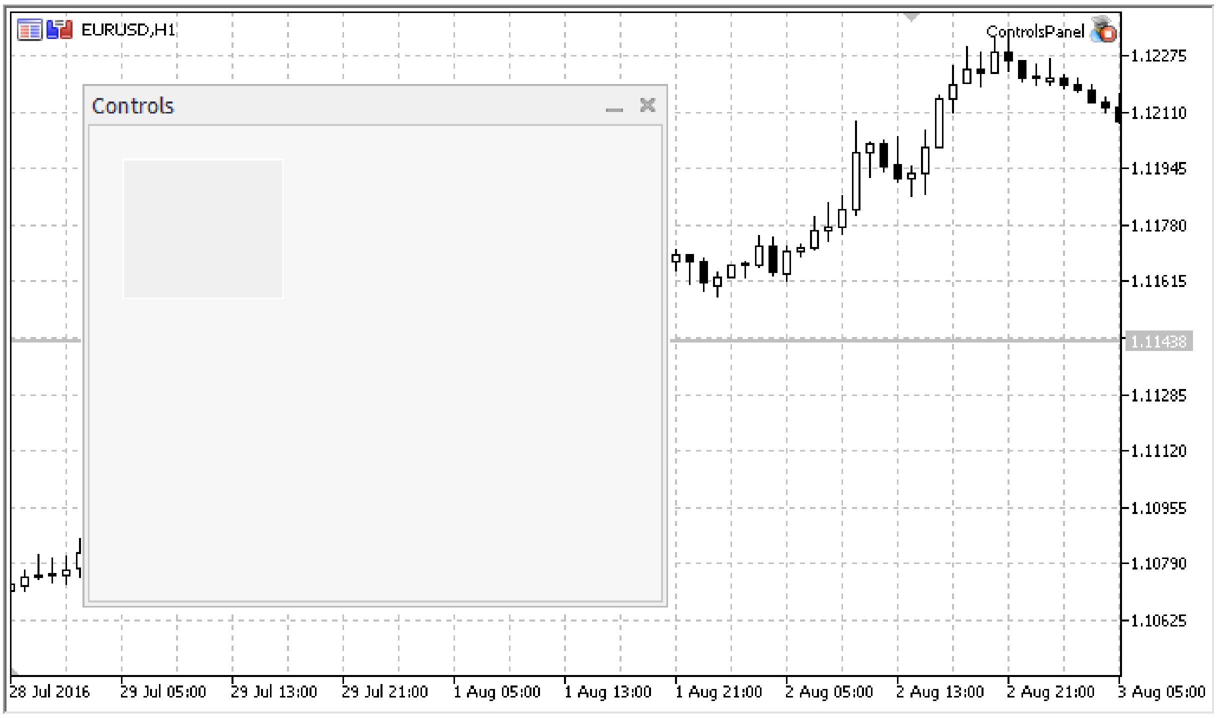
Title

```
   #include <Controls\Panel.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CPanel

Result of the code provided below:


EURUSD,H1
ControlsPanel
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11438
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Chart object properties	
BorderType	Gets the "BorderType" property of the chart object
Chart object event handlers	
OnSetText	"SetText" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
Internal event handlers	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnResize	"Resize" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with Rectangle label:

```
//+------------------------------------------------------------------+
//|                                              ControlsPanel.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPanel"
```

```
#include <Controls\Dialog.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const

protected:
   //--- create dependent controls
   bool              CreatePanel(void);
  };
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
```

```
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreatePanel())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CPanel"                                              |
//+------------------------------------------------------------------+
bool CControlsDialog::CreatePanel(void)
  {
//--- coordinates
   int x1=20;
   int y1=20;
   int x2=ExtDialog.Width()/3;
   int y2=ExtDialog.Height()/3;
//--- create
   if(!my_white_border.Create(0,ExtDialog.Name()+"MyWhiteBorder",m_subwin,x1,y1,x2,y2)
      return(false);
   if(!my_white_border.ColorBackground(CONTROLS_DIALOG_COLOR_BG))
      return(false);
   if(!my_white_border.ColorBorder(CONTROLS_DIALOG_COLOR_BORDER_LIGHT))
      return(false);
   if(!ExtDialog.Add(my_white_border))
      return(false);
   my_white_border.Alignment(WND_ALIGN_CLIENT,0,0,0,0);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//---
CPanel   my_white_border;        // object CPanel
bool     pause=true;             // true - pause
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
```

```
   EventSetTimer(3);
   pause=true;
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
//+------------------------------------------------------------------+
//|  Timer                                                           |
//+------------------------------------------------------------------+
void OnTimer()
  {
   pause=!pause;
  }
```

Create

Creates new CPanel control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

BorderType (Get method)

Gets the "BorderType" property of the chart object.

```
ENUM_BORDER_TYPE  BorderType()
```

Return Value

The value of "BorderType" property.

BorderType (Set method)

Sets new value of "BorderType" property of the chart object.

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type       // value
   )
```

Parameters

type

[in] New value of "BorderType" property.

Return Value

true - successful, otherwise - false.

OnSetText

The virtual handler of control "SetText" (change of the OBJPROP_TEXT property) event.

```
virtual bool  OnSetText()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBackground

The virtual handler of control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnSetColorBorder

The virtual handler of control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event.

```
virtual bool  OnSetColorBackground()
```

Return Value

true - event processed, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

OnResize

The virtual handler of control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

CPicture

CPicture is a class of the simple control based on "Bitmap Label" object.

Description

CPicture class is intended for creation of simple graphic images.

Declaration

```
class CPicture : public CWndObj
```

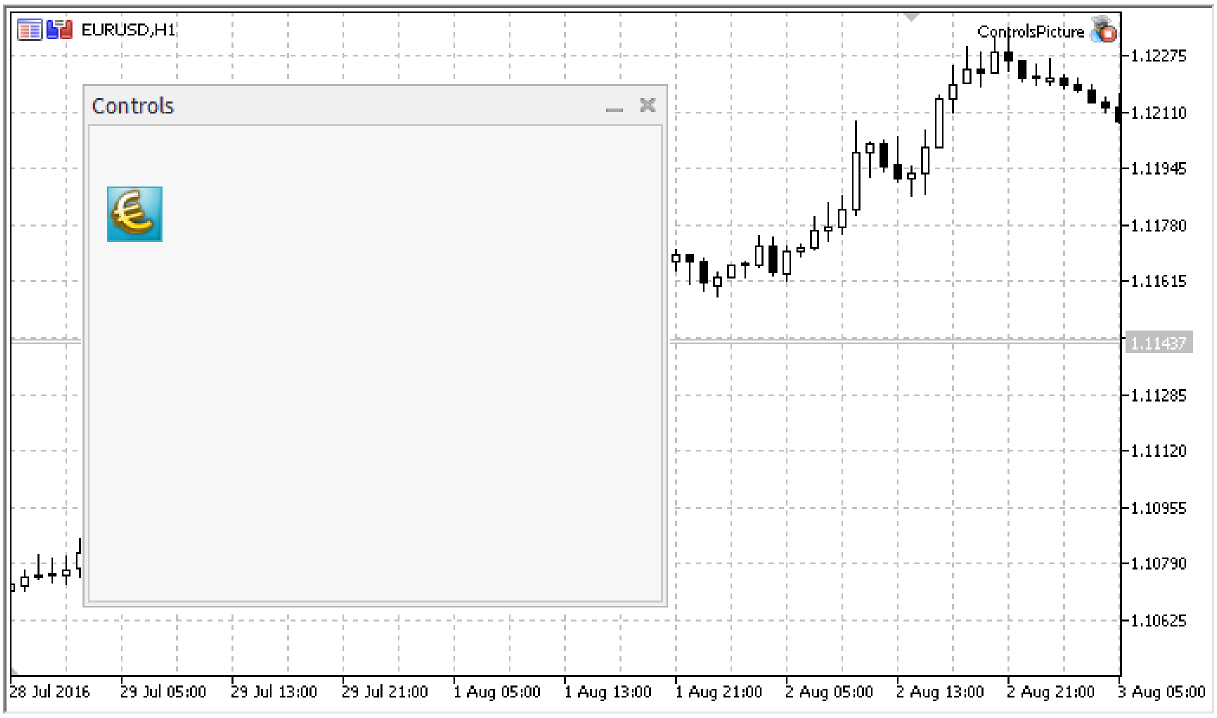
Title

```
#include <Controls\Picture.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndObj
      - CPicture

Result of the code provided below:


EURUSD,H1
ControlsPicture
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11437
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Chart object properties	
Border	Gets/sets the border width of the chart object
BmpName	Gets/sets the name of bmp file of the control
Internal events	
OnCreate	"Create" internal event handler
OnShow	"Show" internal event handler
OnHide	"Hide" internal event handler
OnMove	"Move" internal event handler
OnChange	"Change" internal event handler

Methods inherited from class CObject

Prev, Prev, Next, Next, Save, Load, Type, Compare

Methods inherited from class CWnd

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

Example of creating a panel with Bitmap label:

```
//+------------------------------------------------------------------+
//|                                            ControlsPicture.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CPicture"
#include <Controls\Dialog.mqh>
#include <Controls\Picture.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
```

```
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allc
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CPicture          m_picture;                       // CPicture object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreatePicture(void);
   //--- handlers of the dependent controls events
   void              OnClickPicture(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_picture,OnClickPicture)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
```

```
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreatePicture())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "Picture"                                             |
//+------------------------------------------------------------------+
bool CControlsDialog::CreatePicture(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+32;
   int y2=y1+32;
//--- create
   if(!m_picture.Create(m_chart_id,m_name+"Picture",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set the name of bmp files to display the CPicture control
   m_picture.BmpName("\\Images\\euro.bmp");

   if(!Add(m_picture))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnClickPicture(void)
  {
   Comment(__FUNCTION__);
  }
```

```
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CPicture control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int  Border()   const
```

Return Value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool  Border(
   const int  value      // value
   )
```

Parameters

value

[in] New value of the "Border" property.

Return Value

true - successful, otherwise - false.

BmpName (Get method)

Gets the name of bmp file of the control.

```
string  BmpName()  const
```

Return Value

Name of bmp file of the control.

BmpName (Set method)

Sets the name of bmp file of the control.

```
bool  BmpName(
   const string  name       // file name
   )
```

Parameters

name

[in] Name of bmp file of the control.

Return Value

true - successful, otherwise - false.

OnCreate

The virtual handler of control "Create" internal event.

```
virtual bool  OnCreate()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of control "Show" internal event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnMove

The virtual handler of control "Move" internal event.

```
virtual bool  OnMove()
```

Return Value

true - event processed, otherwise - false.

OnChange

The virtual handler of control "Change" internal event.

```
virtual bool  OnChange()
```

Return Value

true - event processed, otherwise - false.

CScroll

CScroll is a base class for creation of scroll bars.

Description

CScroll is a complex control (with dependent controls), it contains the base functionality for creation of scroll bars. The base class itself is not used as a separate control, two of its heirs (CScrollV and CScrollH classes) are used as controls.

Declaration

```
   class CScroll : public CWndContainer
```

Title

```
   #include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CScroll

Direct descendants

CScrollH, CScrollV

Class Methods by Groups

Create	
Create	Creates control
Chart object event handlers	
OnEvent	Event handler of all chart events
Properties	
MinPos	Gets/sets the minimal position
MaxPos	Gets/sets the maximal position
CurrPos	Gets/sets the current position
Dependent controls creation	
CreateBack	Creates background button
CreateInc	Creates increment button of the scroll bar
CreateDec	Creates decrement button of the scroll bar
CreateThumb	Creates thumb button (can be dragged) of the scroll bar

Create	
Dependent controls event handlers	
OnClickInc	Event handler used for handling increment button events
OnClickDec	Event handler used for handling decrement button events
Internal event handlers	
OnShow	"Create" internal event handler
OnHide	"Hide" internal event handler
OnChangePos	"ChangePosition" internal event handler
Object drag handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

Create

Creates new CScroll control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,           // ID
   const long&      lparam,       // parameter
   const double&    dparam,       // parameter
   const string&    sparam        // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

MinPos (Get method)

Gets the value of "MinPos" (minimal position) of the CScroll control.

```
int  MinPos()  const
```

Return Value

New value of "MinPos" property.

MinPos (Set method)

Sets the value of "MinPos" (minimal position) of the CScroll control.

```
void  MinPos(
   const int  value       // value
   )
```

Parameters

value

[in] New value of "MinPos" property.

Return Value

None.

MaxPos (Get method)

Gets the value of "MaxPos" (maximal position) of the CScroll control.

```
int  MaxPos()  const
```

Return Value

New value of "MaxPos" property.

MaxPos (Set method)

Sets the value of "MaxPos" (maximal position) of the CScroll control.

```
void  MaxPos(
   const int  value      // value
   )
```

Parameters

value

[in] New value of "MaxPos" property.

Return Value

None.

CurrPos (Get method)

Gets the value of "CurrPos" (current position) of the CScroll control.

```
int  CurrPos()  const
```

Return Value

New value of "CurrPos" property.

CurrPos (Set method)

Sets the value of "CurrPos" (current position) of the CScroll control.

```
void  CurrPos(
   const int  value       // value
   )
```

Parameters

value

[in] New value of "CurrPos" property.

Return Value

None.

CreateBack

Creates background button of the CScroll control.

```
virtual bool  CreateBack()
```

Return Value

true - successful, otherwise - false.

CreateInc

Creates increment button of CScroll control.

```
virtual bool  CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of CScroll control.

```
virtual bool  CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of CScroll control.

```
virtual bool  CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnClickInc

The virtual handler of the control "ClickInc" (left mouse button click on the increment button) internal event.

```
virtual bool  OnClickInc()
```

Return Value

true - event processed, otherwise - false.

OnClickDec

The virtual handler of the control "ClickDec" (left mouse button click on the decrement button) internal event.

```
virtual bool  OnClickDec()
```

Return Value

true - event processed, otherwise - false.

OnShow

The virtual handler of the control "Show" event.

```
virtual bool  OnShow()
```

Return Value

true - event processed, otherwise - false.

OnHide

The virtual handler of the control "Hide" internal event.

```
virtual bool  OnHide()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool  OnChangePos()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool  OnThumbDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool  OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int       coord       // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CScrollV

CScrollV is a class of the "Vertical scroll bar" complex control.

Description

CScrollV class is intended for creation of vertical scroll bars.

Declaration

```
class CScrollV : public CScroll
```

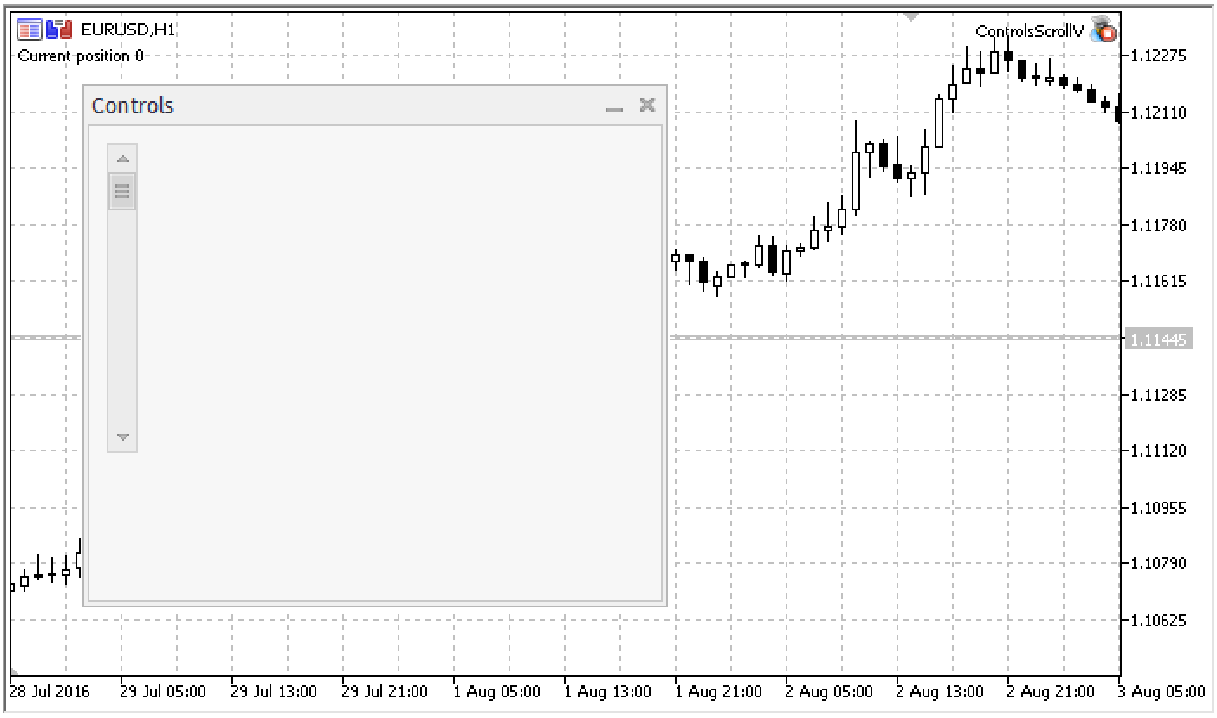
Title

```
#include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CScroll
        - CScrollV

Result of the code provided below:


EURUSD,H1
Current position 0
ControlsScrollV
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11445
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Dependent controls	
CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" internal event handler
OnChangePos	"ChangePosition" internal event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

Methods inherited from class CScroll

Create, OnEvent, MinPos, MinPos, MaxPos, MaxPos, CurrPos, CurrPos

Example of creating a panel with vertical scrollbar:

```
//+------------------------------------------------------------------+
//|                                               ControlsScrollV.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollV"
```

```
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CScrollV          m_scroll_v;                      // CScrollV object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateScrollV(void);
   //--- handlers of the dependent controls events
   void              OnScrollInc(void);
   void              OnScrollDec(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateScrollV())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the CScrollsV object                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateScrollV(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=x1+18;
   int y2=y1+LIST_HEIGHT;
//--- create
   if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollV",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set up the scrollbar
   m_scroll_v.MinPos(0);
//--- set up the scrollbar
   m_scroll_v.MaxPos(10);
   if(!Add(m_scroll_v))
      return(false);
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
```

```
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollInc(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollDec(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
```

```
                   const double& dparam, // event parameter of the double type
                   const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

CreateInc

Creates increment button of the control.

```
virtual bool  CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of the control.

```
virtual bool  CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool  CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool  OnChangePos()
```

Return Value

true - event processed, otherwise - false.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool  OnThumbDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool  OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int      coord      // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CScrollH

CScrollH is a class of the "Horizontal scroll bar" complex control.

Description

CScrollH is intended for creation of horizontal scroll bars.

Declaration

```
class CScrollH : public CScroll
```

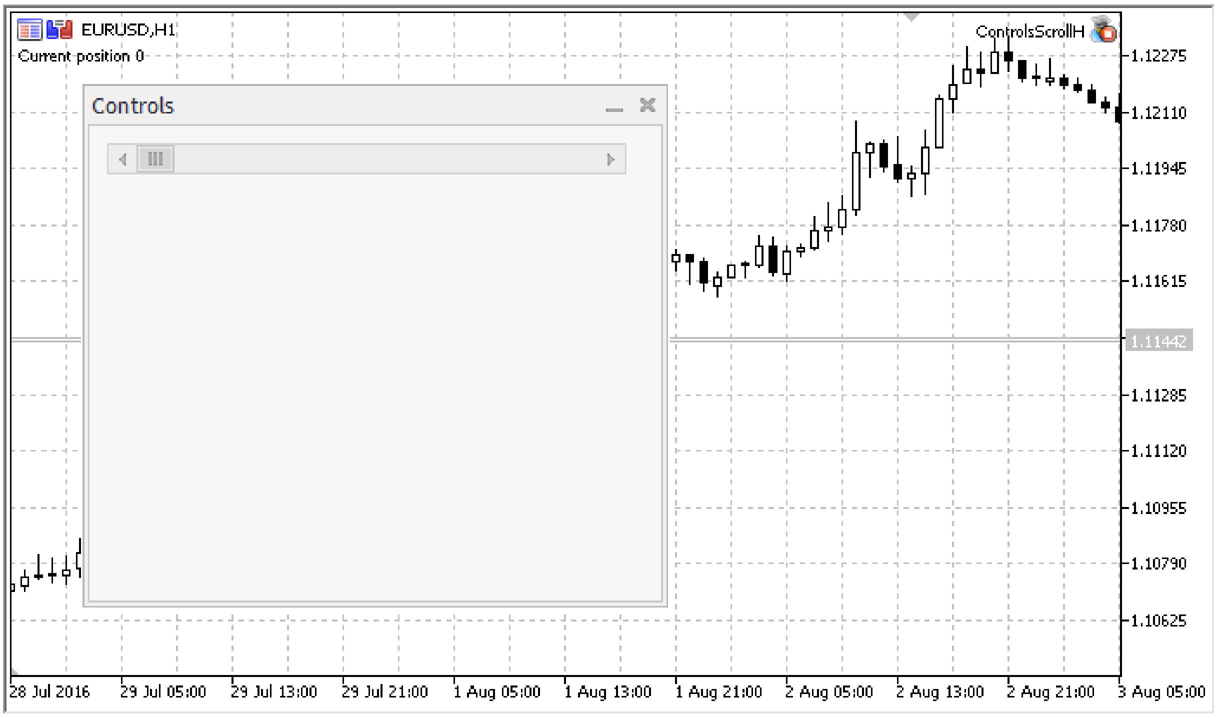
Title

```
#include <Controls\Scrolls.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CScroll
        - CScrollH

Result of the code provided below:


EURUSD,H1
Current position 0
ControlsScrollH
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11442
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Dependent controls	
CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" internal event handler
OnChangePos	"ChangePosition" internal event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar position by coordinate

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

Methods inherited from class CScroll

Create, OnEvent, MinPos, MinPos, MaxPos, MaxPos, CurrPos, CurrPos

Example of creating a panel with horizontal scrollbar:

```
//+------------------------------------------------------------------+
//|                                              ControlsScrollH.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CScrollH"
```

```
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CScrollH          m_scroll_v;                      // CScrollH object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateScrollsH(void);
   //--- handlers of the dependent controls events
   void              OnScrollInc(void);
   void              OnScrollDec(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateScrollsH())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the CScrollsH object                                      |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateScrollsH(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP;
   int x2=x1+3*BUTTON_WIDTH;
   int y2=y1+18;
//--- create
   if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollsH",m_subwin,x1,y1,x2,y2))
      return(false);
//--- set up the scrollbar
   m_scroll_v.MinPos(0);
//--- set up the scrollbar
   m_scroll_v.MaxPos(10);
   if(!Add(m_scroll_v))
      return(false);
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
```

```
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollInc(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnScrollDec(void)
  {
   Comment("Position of the scrollbar ",m_scroll_v.CurrPos());
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
```

```
                    const double& dparam, // event parameter of the double type
                    const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

CreateInc

Creates increment button of the control.

```
virtual bool  CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates decrement button of the control.

```
virtual bool  CreateDec()
```

Return Value

true - successful, otherwise - false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool  CreateThumb()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnChangePos

The virtual handler of the control "ChangePos" (position change) internal event.

```
virtual bool  OnChangePos()
```

Return Value

true - event processed, otherwise - false.

OnThumbDragStart

The virtual handler of the control "ThumbDragStart" (drag start) event.

```
virtual bool  OnThumbDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The virtual handler of the control "ThumbDragProcess" event.

```
virtual bool  OnThumbDragProcess(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The virtual handler of the control "ThumbDragEnd" (drag process finished) event.

```
virtual bool  OnThumbDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets the control scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int       coord       // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Return Value

Scroll bar position.

CWndClient

CWndClient is a class of the "Client area" complex control (with dependent controls). It is a base class for creation of scroll bars area.

Description

CWndClient implements the functionality for creation of client area with scroll bars.

Declaration

```
class CWndClient : public CWndContainer
```

Title

```
#include <Controls\WndClient.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CWndClient

Direct descendants

CCheckGroup, CListView, CRadioGroup

Class Methods by Groups

Create	
Create	Creates control
Chart event handler	
OnEvent	Event handler of all chart events
Properties	
ColorBackground	Sets background color
ColorBorder	Sets border color
BorderType	Sets border type
Settings	
VScrolled	Gets/sets the flag indicating that vertical scroll bar is used
HScrolled	Gets/sets the flag indicating that horizontal scroll bar is used
Dependent controls	
CreateBack	Creates background for scroll bar

Create	
CreateScrollV	Creates vertical scroll bar
CreateScrollH	Creates horizontal scroll bar
Internal event handlers	
OnResize	"Resize" internal event handler
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnHScrollShow	"Show" internal event handler (virtual) of HScroll dependent control
OnHScrollHide	"Hide" internal event handler (virtual) of HScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnScrollLineLeft	"ScrollLineLeft" internal event handler (virtual) of HScroll dependent control
OnScrollLineRight	"ScrollLineRight" internal event handler (virtual) of HScroll dependent control
Resize	
Rebound	Sets new parameters of the control using CRect class coordinates

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide, Save, Load

Create

Creates new CWndClient control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

ColorBackground

Sets background color of the control.

```
bool  ColorBackground(
   const color  value      // background color
   )
```

Parameters

value

[in] Background color of the control.

Return Value

true - successful, otherwise - false.

ColorBorder

Set border color of the control.

```
bool  ColorBorder(
   const color  value      // color
   )
```

Parameters

value

[in] Border color of the control.

Return Value

true - successful, otherwise - false.

BorderType

Sets border type of the control.

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type      // value
   )
```

Parameters

type

[in] Border type of the control.

Return Value

true - successful, otherwise - false.

VScrolled (Get method)

Gets the flag indicating that vertical scroll bar is used.

```
bool  VScrolled()
```

Return Value

true - vertical scroll bar is used, otherwise - false.

VScrolled (Set method)

Sets the flag indicating that vertical scroll bar is used.

```
bool  VScrolled(
   const bool  flag       // flag
   )
```

Parameters

flag

[in] Flag.

Return Value

true - successful, otherwise - false.

HScrolled (Get method)

Gets the flag indicating that horizontal scroll bar is used.

```
bool  HScrolled()
```

Return Value

true - horizontal scroll bar is used, otherwise - false.

HScrolled (Set method)

Sets the flag indicating that horizontal scroll bar is used.

```
bool  HScrolled(
   const bool   flag       // flag
   )
```

Parameters

flag

[in] Flag.

Return Value

true - successful, otherwise - false.

CreateBack

Creates background button of the control.

```
virtual bool  CreateBack()
```

Return Value

true - successful, otherwise - false.

CreateScrollV

Creates vertical scroll bar.

```
virtual bool  CreateScrollV()
```

Return Value

true - successful, otherwise - false.

CreateScrollH

Creates horizontal scroll bar.

```
virtual bool  CreateScrollH()
```

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnVScrollShow

The virtual handler of the VScroll (vertical scroll) dependent control "Show" (vertical scroll bar show) internal event.

```
virtual bool  OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnVScrollHide

The virtual handler of the VScroll (vertical scroll) dependent control "Hide" (vertical scroll bar hide) internal event.

```
virtual bool  OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHScrollShow

The virtual handler of the HScroll (horizontal scroll) dependent control "Show" (horizontal scroll bar show) internal event.

```
virtual bool  OnHScrollShow()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnHScrollHide

The virtual handler of the HScroll (horizontal scroll) dependent control "Hide" (horizontal scroll bar hide) internal event.

```
virtual bool  OnHScrollHide()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineDown

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineDown" (vertical scroll line down) internal event.

```
virtual bool  OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineUp

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineUp" (vertical scroll line up) internal event.

```
virtual bool  OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineLeft

The virtual handler of the HScroll (horizontal scroll) dependent control "ScrollLineLeft" (horizontal scroll line left) internal event.

```
virtual bool  OnScrollLineLeft()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

OnScrollLineRight

The virtual handler of the HScroll (horizontal scroll) dependent control "ScrollLineRight" (horizontal scroll line right) internal event.

```
virtual bool  OnScrollLineRight()
```

Return Value

true - event processed, otherwise - false.

Note

The base class method does nothing and always returns true.

ReBound

Sets new parameters of the control using CRect class coordinates.

```
void  ReBound(
   const & CRect   rect       // CRect class
   )
```

Return Value

None.

CListView

CListView is a class of the ListView complex control (with dependent controls).

Description

CListView class encapsulates list-control functionality.

Declaration

```
class CListView : public CWndClient
```

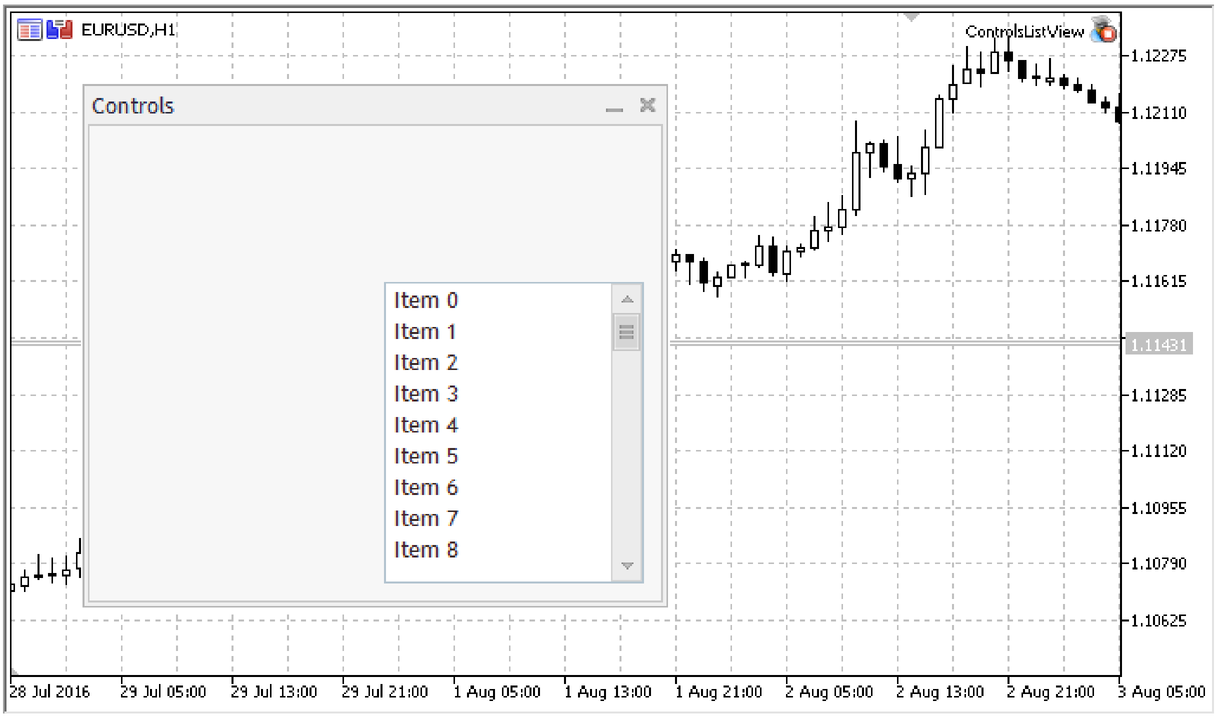
Title

```
#include <Controls\ListView.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CWndClient
        - CListView

Result of the code provided below:


EURUSD,H1
ControlsListView
Controls
Item 0
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
1.12275
1.12110
1.11945
1.11780
1.11615
1.11431
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Settings	
TotalView	Sets the number of items shown on the control
Add/Delete	
AddItem	Adds an item to control list
Data	
Select	Selects current list element by index
SelectByText	Selects current list element by text
SelectByValue	Selects current list element by value
Read-only data	
Value	Gets the value of the current list element
Dependent controls	
CreateRow	Creates a row of the ListView
Internal event handlers	
OnResize	"Resize" internal event handler (virtual)
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnItemClick	"ItemClick" internal event handler (virtual)
Redraw	
Redraw	Redraws the control
RowState	Sets the state of the specified row
CheckView	Checks the "visibility" of the specified row

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide, Save, Load

Methods inherited from class CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

Example of creating a panel with list view control:

```
//+------------------------------------------------------------------+
//|                                             ControlsListView.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CListView"
#include <Controls\Dialog.mqh>
#include <Controls\ListView.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
```

```
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CListView         m_list_view;                     // CListView object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateListView(void);
   //--- handlers of the dependent controls events
   void              OnChangeListView(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_list_view,OnChangeListView)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
```

```
      return(false);
//--- create dependent controls
   if(!CreateListView())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "ListView" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateListView(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+GROUP_WIDTH+2*CONTROLS_GAP_X;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+2*CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+LIST_HEIGHT-CONTROLS_GAP_Y;
//--- create
   if(!m_list_view.Create(m_chart_id,m_name+"ListView",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_list_view))
      return(false);
//--- fill out with strings
   for(int i=0;i<16;i++)
      if(!m_list_view.AddItem("Item "+IntegerToString(i)))
         return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeListView(void)
  {
   Comment(__FUNCTION__+" \""+m_list_view.Select()+"\"");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
```

```
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates CListView control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

TotalView

Sets the number of items shown on the control.

```
bool  TotalView(
   const int   value       // number of items shown
   )
```

Parameters

value

[in] The number of items shown on the control.

Return Value

true - successful, otherwise - false.

Note

The number of shown items can be specified only at once.

AddItem

Adds an item to the control list.

```
bool  AddItem(
   const string  item,      // text
   const long    value      // value
   )
```

Parameters

item

[in] Text.

value

[in] Value.

Return Value

true - successful, otherwise - false.

Select

Selects current list element by index.

```
bool  Select(
   const int  index      // index
   )
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

SelectByText

Selects the current list element by text.

```
bool  SelectByText(
   const string  text      // text
   )
```

Parameters

text

[in] Element text.

Return Value

true - successful, otherwise - false.

SelectByValue

Selects the current list element by value.

```
bool  SelectByValue(
   const long  value      // value
   )
```

Parameters

value

[in] Value.

Return Value

true - successful, otherwise - false.

Value

Gets the value of the current list element.

```
long  Value()
```

Return Value

The value of the current list element.

CreateRow

Creates a row of the CListView control.

```
bool  CreateRow(
   const int  index      // index
   )
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

OnResize

The virtual handler of the control "Resize" internal event.

```
virtual bool  OnResize()
```

Return Value

true - event processed, otherwise - false.

OnVScrollShow

The virtual handler of the VScroll (vertical scroll) dependent control "VScrollShow" (vertical scroll bar show) internal event.

```
virtual bool  OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the VScroll (vertical scroll) dependent control "VScrollHide" (vertical scroll bar hide) internal event.

```
virtual bool  OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineDown" (vertical scroll line down) internal event.

```
virtual bool  OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the VScroll (vertical scroll) dependent control "ScrollLineUp" (vertical scroll line up) internal event.

```
virtual bool  OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnItemClick

The virtual handler of "ItemClick" (mouse button click) internal event on a specified row of CListView control.

```
virtual bool  OnItemClick()
   const int     index      // index
   )
```

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Changes the state of the specified row of the CListView control.

```
bool  RowState(
   const int    index       // index
   const bool   select      // state
   )
```

Parameters

index

[in] Row index.

select

[in] Row state.

Return Value

true - successful, otherwise - false.

CheckView

Checks the "visibility" of the specified row of the CListView control.

```
bool  CheckView()
```

Return Value

true - selected row is visible, otherwise - false.

CComboBox

CComboBox is a class of the ComboBox complex control (with dependent controls).

Description

ComboBox consists of a list box, combined with a static control, intended for selection. The list-box portion of the control may be dropped down when a user selects the drop-down arrow next to the control

Declaration

```
class CComboBox : public CWndContainer
```

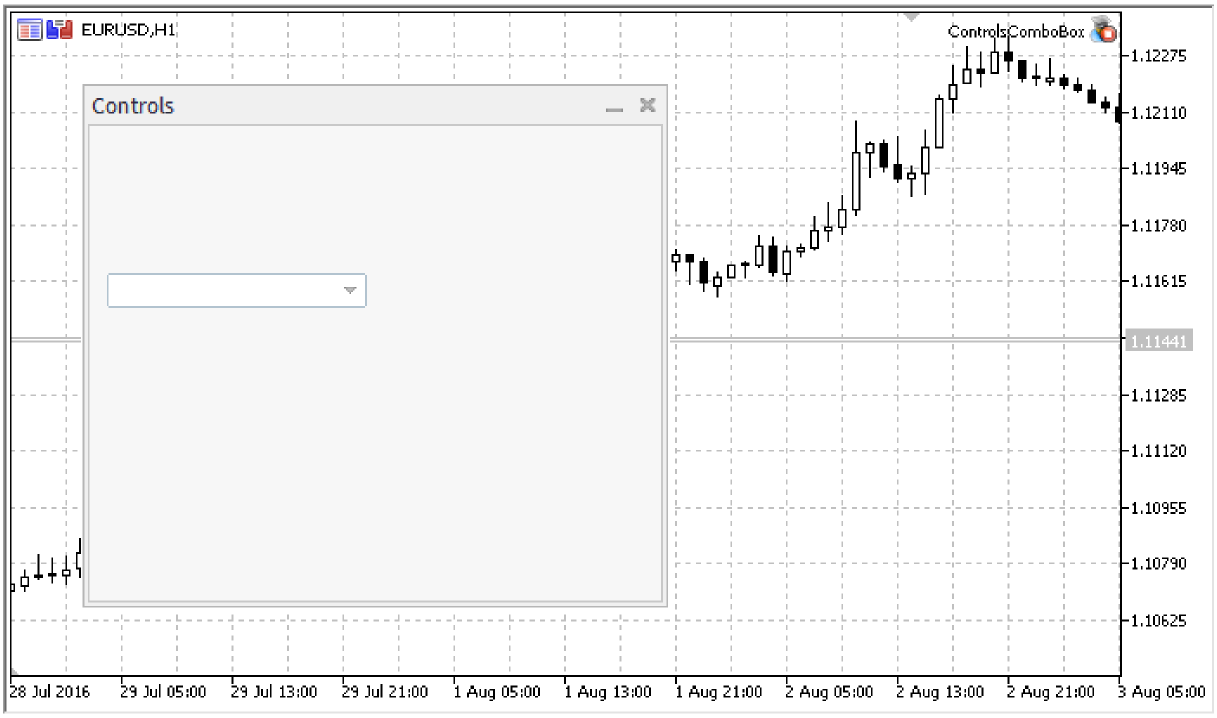
Title

```
#include <Controls\ComboBox.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CComboBox

Result of the code provided below:


EURUSD,H1
ControlsComboBox
Controls
1.12275
1.12110
1.11945
1.11780
1.11615
1.11441
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds an item to control
Settings	
ListViewItems	Sets the number of items shown on the control
Data	
Select	Selects the current list element by index
SelectByText	Selects the current list element by text
SelectByValue	Selects the current list element by value
Read-only data	
Value	Gets the value of the current list element
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateButton	Creates dependent control (button)
CreateList	Creates dependent control (list view)
Dependent controls event handlers	
OnClickEdit	"ClickEdit" internal event handler (virtual)
OnClickButton	"ClickButton" internal event handler (virtual)
OnChangeList	"ChangeList" internal event handler (virtual)
Show/Hide the drop-down list	
ListShow	Shows the items list
ListHide	Hides the items list

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet,

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Hide

Example of creating a panel with Combobox control:

```
//+------------------------------------------------------------------+
//|                                             ControlsComboBox.mq5 |
//|                        Copyright 2015, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2015, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CComboBox"
#include <Controls\Dialog.mqh>
#include <Controls\ComboBox.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
```

```
private:
   CComboBox         m_combo_box;;                     // CComboBox object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateComboBox(void);
   //--- handlers of the dependent controls events
   void              OnChangeComboBox(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_combo_box,OnChangeComboBox)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateComboBox())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "ComboBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateComboBox(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_combo_box.Create(m_chart_id,m_name+"ComboBox",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_combo_box))
      return(false);
//--- fill out with strings
   for(int i=0;i<16;i++)
      if(!m_combo_box.ItemAdd("Item "+IntegerToString(i)))
         return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeComboBox(void)
  {
   Comment(__FUNCTION__+" \""+m_combo_box.Select()+"\"");
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates CComboBox control.

```
virtual bool  Create(
   const long    chart,       // chart ID
   const string  name,        // name
   const int     subwin,      // chart subwindow
   const int     x1,          // coordinate
   const int     y1,          // coordinate
   const int     x2,          // coordinate
   const int     y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chat, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value      // value
   )
```

Parameters

item

[in] Text.

value=0

[in] Value.

Return Value

true - successful, otherwise - false.

ListViewItems

Sets the number of list elements of the CComboBox control.

```
void  ListViewItems(
   const int     value      // number of elements
   )
```

Parameters

value

[in] Number of the drop-down list elements.

Return Value

true - successful, otherwise - false.

Select

Selects the current list element by index.

```
bool  Select(
   const int  index      // index
   )
```

Parameters

index

[in] Item index.

Return Value

true - successful, otherwise - false.

SelectByText

Selects the current list element by a specified text.

```
bool  SelectByText(
   const string  text      // text
   )
```

Parameters

text

[in] Text of the item.

Return Value

true - successful, otherwise - false.

SelectByValue

Selects the current list element by a specified value.

```
bool  SelectByValue(
   const long  value      // value
   )
```

Parameters

value

[in] Value.

Return Value

true - successful, otherwise - false.

Value

Gets the value of the current list element.

```
long  Value()
```

Return Value

The value of the current list element.

CreateEdit

Creates dependent control (edit) of the control.

```
virtual bool  CreateEdit()
```

Return Value

true - successful, otherwise - false.

CreateButton

Creates dependent control (button).

```
virtual bool  CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateList

Creates dependent control (list view).

```
virtual bool  CreateList()
```

Return Value

true - successful, otherwise - false.

OnClickEdit

The virtual handler of the control "ClickEdit" (mouse click on the edit) internal event.

```
virtual bool  OnClickEdit()
```

Return Value

true - event processed, otherwise - false.

OnClickButton

The virtual handler of the "ClickButton" (mouse click on the button) internal event.

```
virtual bool  OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnChangeList

The virtual handler of the "ChangeList" (change of the list) internal event.

```
virtual bool  OnChangeList()
```

Return Value

true - event processed, otherwise - false.

ListShow

Shows the drop-down list of the control.

```
virtual bool  ListShow()
```

Return Value

true - successful, otherwise - false.

ListHide

Hides the drop-down list of the control.

```
virtual bool  ListHide()
```

Return Value

true - successful, otherwise - false.

CCheckBox

CCheckBox is class of the CheckBox complex control.

Description

CCheckBox control displays a check box that allows a user to select a true or false condition at a mouse click.

Declaration

```
class CCheckBox : public CWndContainer
```

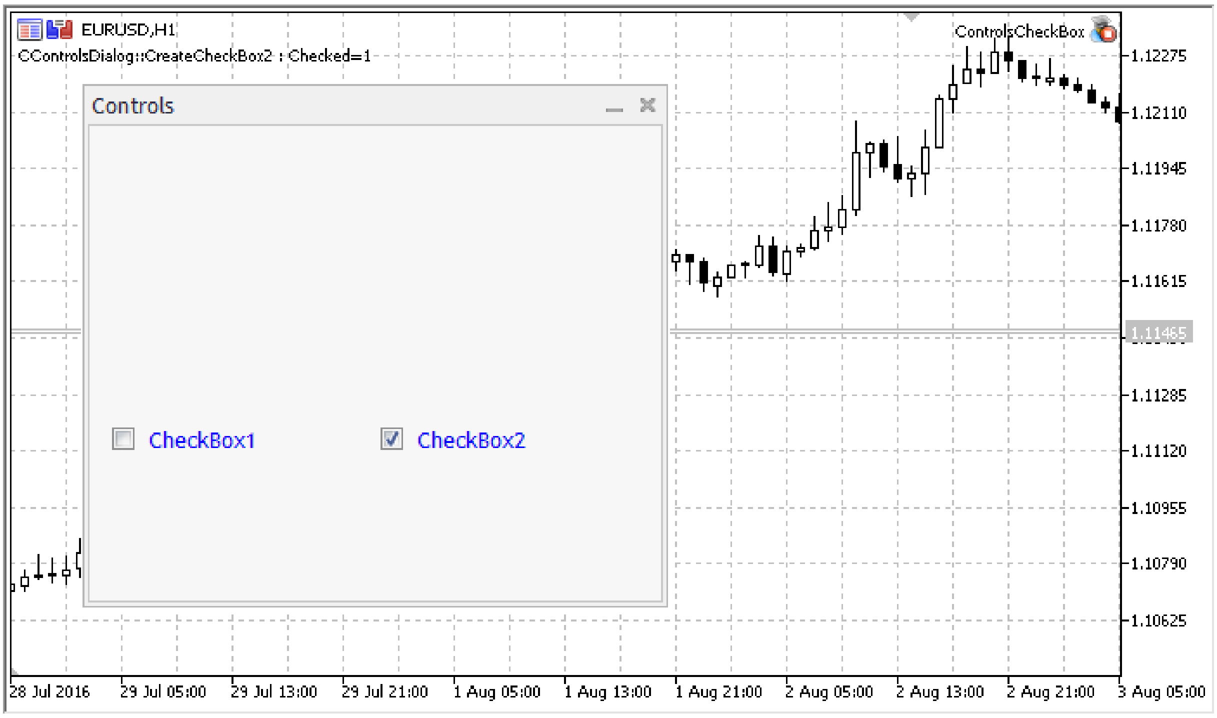
Title

```
#include <Controls\CheckBox.mqh>
```

Inheritance hierarchy

- [CObject](#)
  - [CWnd](#)
    - [CWndContainer](#)
      - CCheckBox

Result of the [code](#) provided below:



CControlsDialog::CreateCheckBox2 : Checked=1

EURUSD,H1
ControlsCheckBox
Controls
☐ CheckBox1
☑ CheckBox2
1.12275
1.12110
1.11945
1.11780
1.11615
1.11465
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
[Create](#)	Creates control

Create	
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/sets text label associated with the control
Color	Gets/sets color of text label associated with the control
State	
Checked	Gets/sets a value indicating whether the control is checked
Data	
Value	Gets/sets the value associated with the control
Dependent controls	
CreateButton	Creates dependent control (button)
CreateLabel	Creates dependent control (label)
Dependent controls event handlers	
ClickButton	"ClickButton" internal event handler (virtual)
ClickLabel	"ClickLabel" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

Example of creating a panel with Checkbox control:

```
//+------------------------------------------------------------------+
//|                                              ControlsCheckBox.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
```

```
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckBox"
#include <Controls\Dialog.mqh>
#include <Controls\CheckBox.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allov
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allc
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CCheckBox         m_check_box1;                    // CCheckBox object
   CCheckBox         m_check_box2;                    // CCheckBox object
public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons
protected:
   //--- create dependent controls
   bool              CreateCheckBox1(void);
   bool              CreateCheckBox2(void);
   //--- handlers of the dependent controls events
   void              OnChangeCheckBox1(void);
   void              OnChangeCheckBox2(void);
  };
```

```
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_box1,OnChangeCheckBox1)
ON_EVENT(ON_CHANGE,m_check_box2,OnChangeCheckBox2)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateCheckBox1())
      return(false);
   if(!CreateCheckBox2())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckBox1(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
```

```
   if(!m_check_box1.Create(m_chart_id,m_name+"CheckBox1",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_check_box1.Text("CheckBox1"))
      return(false);
   if(!m_check_box1.Color(clrBlue))
      return(false);
   if(!Add(m_check_box1))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckBox" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckBox2(void)
  {
//--- coordinates
   int x1=INDENT_LEFT+GROUP_WIDTH+CONTROLS_GAP_X;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+BUTTON_HEIGHT;
//--- create
   if(!m_check_box2.Create(m_chart_id,m_name+"CheckBox2",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!m_check_box2.Text("CheckBox2"))
      return(false);
   if(!m_check_box2.Color(clrBlue))
      return(false);
   if(!Add(m_check_box2))
      return(false);
   m_check_box2.Checked(true);
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckBox1(void)
  {
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box1.Checked()));
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckBox2(void)
  {
   Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID  
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates CCheckBox control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets text of the label associated with the control.

```
string  Text()
```

Return Value

Text of the label.

Text (Set method)

Sets text of the label associated with the control.

```
bool  Text(
   const string  value       // text
   )
```

Parameters

value

[in] New text of the label.

Return Value

true - successful, otherwise - false.

Note

Label text is specified by setting OBJPROP_TEXT (text) of a chart object.

Color (Get method)

Gets color of the label associated with the control.

```
color  Color()  const
```

Return Value

Label color.

Color (Set method)

Sets color of the label associated with the control.

```
bool  Color(
   const color  value      // color
   )
```

Parameters

value

[in] New label color.

Return Value

true - successful, otherwise - false.

Note

Label color is specified by setting OBJPROP_COLOR (color) of a chart object.

Checked (Get method)

Gets state of the control.

```
bool  Checked()   const
```

Return Value

State of the control.

Checked (Set method)

Sets state of the control.

```
bool  Checked(
   const bool   flag       // state
   )
```

Parameters

flag

[in] New state.

Return Value

true - successful, otherwise - false.

Value (Get method)

Gets the value associated with the control.

```
int  Value()  const
```

Return Value

The value associated with the control.

Value (Set method)

Sets the value associated with the control.

```
void  Value(
   const int  value       // value
   )
```

Parameters

value

[in] New value.

Return Value

None.

CreateButton

Creates dependent control (button).

```
virtual bool  CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateLabel

Creates dependent control (label).

```
virtual bool  CreateLabel()
```

Return Value

true - successful, otherwise - false.

OnClickButton

The virtual handler of the control "ClickButton" (mouse click on the button) internal event.

```
virtual bool  OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnClickLabel

The virtual handler of the control "ClickLabel" (mouse click on the label) internal event.

```
virtual bool  OnClickLabel()
```

Return Value

true - event processed, otherwise - false.

CCheckGroup

CCheckGroup is a class of the CheckGroup complex control (with dependent controls).

Description

CCheckGroup provides the possibility for creation of controls, which allow to display and edit flags.

Declaration

```
   class CCheckGroup : public CWndClient
```

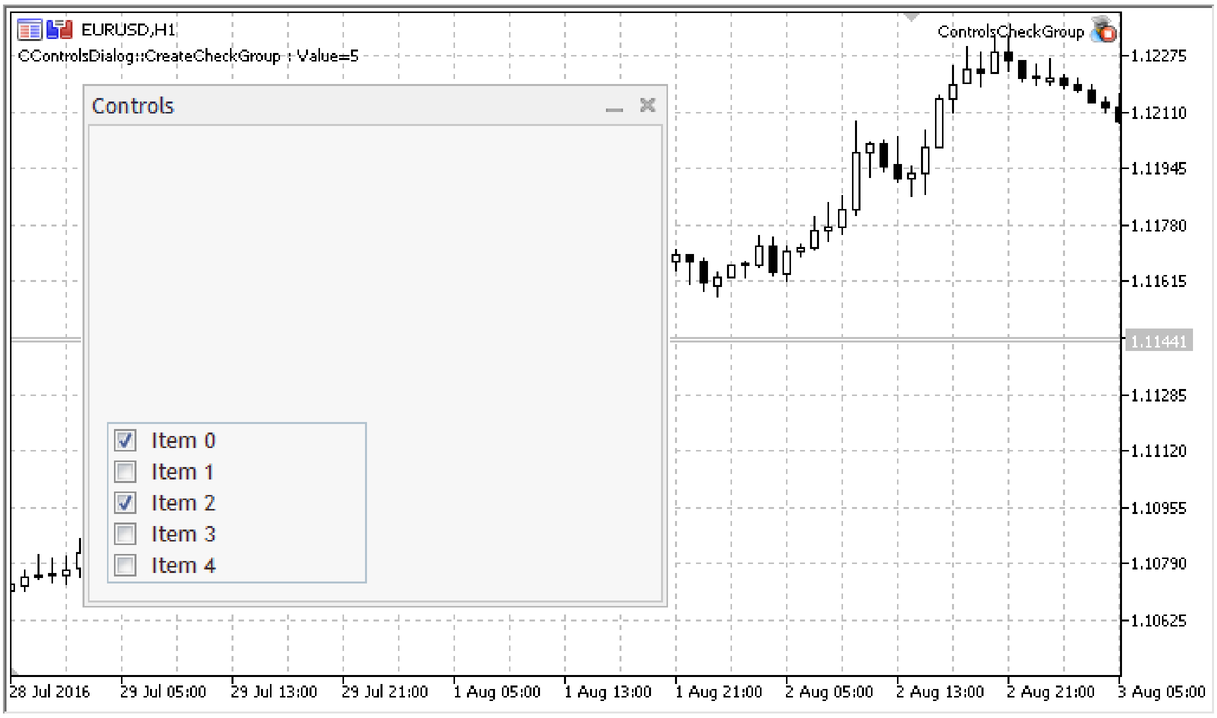
Title

```
   #include <Controls\CheckGroup.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CWndClient
        - CCheckGroup

Result of the code provided below:



EURUSD,H1

CControlsDialog::CreateCheckGroup : Value=5

Controls

☑ Item 0
☐ Item 1
☑ Item 2
☐ Item 3
☐ Item 4

ControlsCheckGroup
1.12275
1.12110
1.11945
1.11780
1.11615
1.11441
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value associated with the control
Dependent controls	
CreateButton	Creates new CCheckBox item at a specified index
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" internal event handler (virtual) of VScroll dependent control
Redraw	
Redraw	Redraws the group
RowState	Sets the state of the specified item of the group

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide

Methods inherited from class CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

Example of creating a panel with Checkbox group control:

```
//+------------------------------------------------------------------+
//|                                           ControlsCheckGroup.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckGroup"
#include <Controls\Dialog.mqh>
#include <Controls\CheckGroup.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CCheckGroup       m_check_group;                   // CCheckGroup object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateCheckGroup(void);
   //--- handlers of the dependent controls events
   void              OnChangeCheckGroup(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_group,OnChangeCheckGroup)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateCheckGroup())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "CheckGroup" element                                  |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateCheckGroup(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (RADIO_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+CHECK_HEIGHT;
//--- create
   if(!m_check_group.Create(m_chart_id,m_name+"CheckGroup",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_check_group))
      return(false);
//--- fill out with strings
   for(int i=0;i<5;i++)
      if(!m_check_group.AddItem("Item "+IntegerToString(i),1<<i))
         return(false);
   m_check_group.Check(0,1<<0);
   m_check_group.Check(2,1<<2);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeCheckGroup(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(ChartID(),"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
```

```
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CCheckGroup control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item (row) to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value      // value
   )
```

Parameters

item

[in] Added control label text.

value=0

[in] Value associated with the added control.

Return Value

true - successful, otherwise - false.

Value

Gets the value associated with the control.

```
long  Value()
```

Return Value

The value associated with the control.

Note

The value depends on state of all items of the CCheckGroup.

CreateButton

Creates new CCheckBox class instance at a specified index.

```
bool  CreateButton(
   int   index       // index
   )
```

Parameters

index

[in] Index of the new item in the CCheckGroup.

Return Value

true - successful, otherwise - false.

OnVScrollShow

The virtual handler of the dependent VScroll (vertical scroll) control "Show" internal event.

```
virtual bool  OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the dependent VScroll (vertical scroll) control "Hide" internal event.

```
virtual bool  OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineDown" internal event.

```
virtual bool  OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineUp" internal event.

```
virtual bool  OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnChangeItem

The virtual handler of the control "ChangeItem" (item change) internal event.

```
virtual bool  OnChangeItem(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the changed item.

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Sets the state of the specified item.

```
bool  RowState(
   const int    index,       // item index
   const bool   select       // state
   )
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Return Value

true - successful, otherwise - false.

CRadioButton

CRadioButton is a class of RadioButton complex control.

Description

CRadioButton itself is not used, it used for creation of CRadioGroup items.

Declaration

```
class CRadioButton : public CWndContainer
```

Title

```
#include <Controls\RadioButton.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CRadioButton

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Properties	
Text	Gets/sets the text label associated with the control
Color	Gets/sets the color of text label associated with the control
State	
State	Gets/sets the state
Dependent controls	
CreateButton	Creates button
CreateLabel	Creates label
Dependent controls event handlers	
OnClickButton	"ClickButton" internal event handler (virtual)
OnClickLabel	"ClickLabel" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide, Save, Load

Create

Creates new CRadioButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

Text (Get method)

Gets text of the label associated with the control.

```
string  Text()
```

Return Value

Text of the label.

Text (Set method)

Sets text of the label.

```
bool  Text(
   const string  value      // text
   )
```

Parameters

value

[in] New text of the label.

Return Value

true - successful, otherwise - false.

Color (Get method)

Gets color of the label associated with the control.

```
color  Color()  const
```

Return Value

Label color.

Color (Set method)

Sets color of the label associated with the control.

```
bool  Color(
   const color  value      // color
   )
```

Parameters

value

[in] New label color.

Return Value

true - successful, otherwise - false.

State (Get method)

Gets the button state.

```
bool  State()  const
```

Return Value

Button state.

State (Set method)

Sets the button state.

```
bool  State(
   const bool  flag      // flag
   )
```

Parameters

flag

[in] New button state.

Return Value

true - successful, otherwise - false.

CreateButton

Creates button.

```
virtual bool  CreateButton()
```

Return Value

true - successful, otherwise - false.

CreateLabel

Creates label.

```
virtual bool  CreateLabel()
```

Return Value

true - successful, otherwise - false.

OnClickButton

The virtual handler of the control "ClickButton" (button click) internal event.

```
virtual bool  OnClickButton()
```

Return Value

true - event processed, otherwise - false.

OnClickLabel

The virtual handler of the control "ClickLabel" (click on the label) internal event.

```
virtual bool  OnClickLabel()
```

Return Value

true - event processed, otherwise - false.

CRadioGroup

CRadioGroup is a class of RadioGroup complex control (with dependent controls).

Description

CRadioGroup enables creation of a control allowing the display and editing the enumerable type field.

Declaration

```
class CRadioGroup : public CWndClient
```

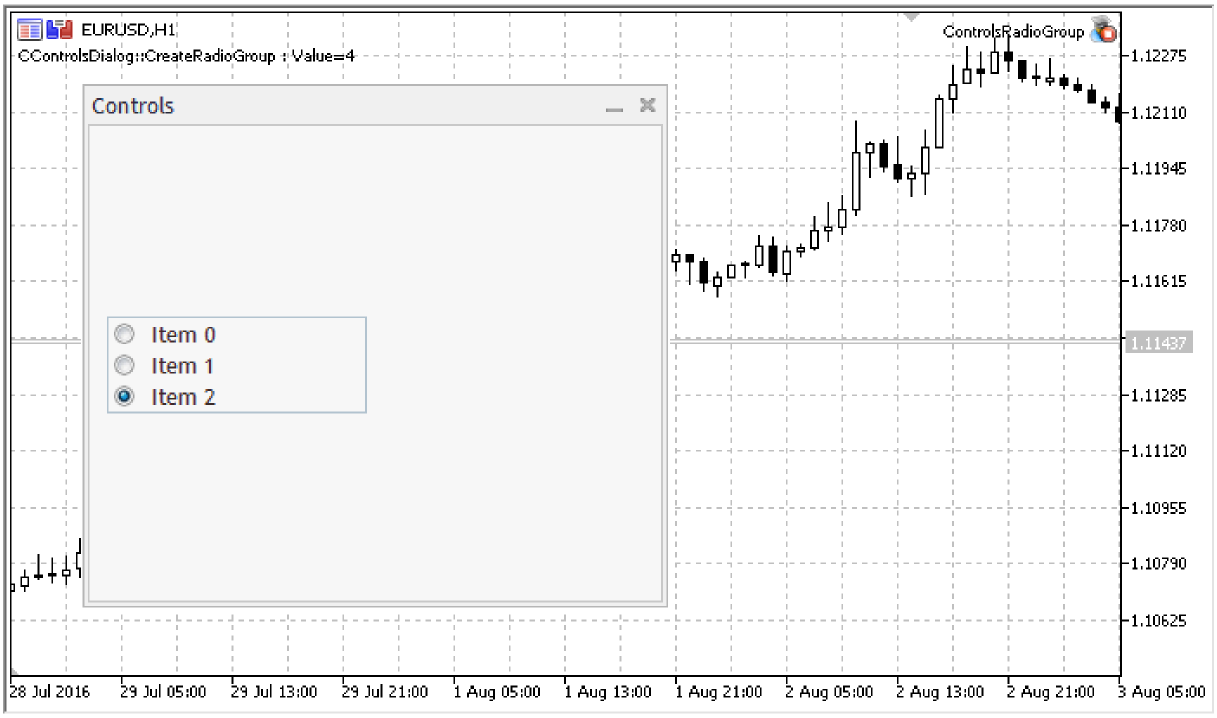
Title

```
#include <Controls\RadioGroup.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CWndClient
        - CRadioGroup

Result of the code provided below:



EURUSD,H1

CControlsDialog::CreateRadioGroup : Value=4

ControlsRadioGroup
Controls
○ Item 0
○ Item 1
● Item 2
1.12275
1.12110
1.11945
1.11780
1.11615
1.11437
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value associated with the control
Dependent controls	
CreateButton	Creates new CRadioButton item at a specified index
Dependent controls event handlers	
OnVScrollShow	"Show" internal event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" internal event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" internal event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" internal event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" internal event handler (virtual)
Redraw	
Redraw	Redraws the group of items
RowState	Sets the state of the specified item
Select	Selects the current item

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Enable, Disable, Hide

Methods inherited from class CWndClient

ColorBackground, ColorBorder, BorderType, VScrolled, VScrolled, HScrolled, HScrolled, Id

Example of creating a panel with group of radio buttons:

```
//+------------------------------------------------------------------+
//|                                             ControlsRadioGroup.mq5 |
//|                         Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CRadioGroup"
#include <Controls\Dialog.mqh>
#include <Controls\RadioGroup.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CRadioGroup       m_radio_group;                   // CRadioGroup object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateRadioGroup(void);
   //--- handlers of the dependent controls events
   void              OnChangeRadioGroup(void);
  };
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_radio_group,OnChangeRadioGroup)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateRadioGroup())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "RadioGroup" element                                  |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateRadioGroup(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (BUTTON_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y)+
          (EDIT_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+RADIO_HEIGHT;
//--- create
   if(!m_radio_group.Create(m_chart_id,m_name+"RadioGroup",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_radio_group))
      return(false);
//--- fill out with strings
   for(int i=0;i<3;i++)
      if(!m_radio_group.AddItem("Item "+IntegerToString(i),1<<i))
         return(false);
   m_radio_group.Value(1<<2);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeRadioGroup(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID  
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CRadioGroup control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value=0    // value
   )
```

Parameters

item

[in] Added control label text.

value=0

[in] Value associated with the added control.

Return Value

true - successful, otherwise - false.

Value

Gets the value associated with the control.

```
long  Value()
```

Return Value

The value associated with the control.

Note

The value depends on state of all CRadioButton items of the CRadioGroup control.

CreateButton

Creates new CRadioButton class instance at a specified index.

```
bool  CreateButton(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the new item in the CRadioGroup.

Return Value

true - successful, otherwise - false.

OnVScrollShow

The virtual handler of the dependent VScroll (vertical scroll) control "Show" internal event.

```
virtual bool  OnVScrollShow()
```

Return Value

true - event processed, otherwise - false.

OnVScrollHide

The virtual handler of the dependent VScroll (vertical scroll) control "Hide" internal event.

```
virtual bool  OnVScrollHide()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineDown

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineDown" internal event.

```
virtual bool  OnScrollLineDown()
```

Return Value

true - event processed, otherwise - false.

OnScrollLineUp

The virtual handler of the dependent VScroll (vertical scroll) control "ScrollLineUp" internal event.

```
virtual bool  OnScrollLineUp()
```

Return Value

true - event processed, otherwise - false.

OnChangeItem

The virtual handler of the control "ChangeItem" (item change) event.

```
virtual bool  OnChangeItem(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the changed item.

Return Value

true - event processed, otherwise - false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Return Value

true - successful, otherwise - false.

RowState

Changes the state of an item.

```
bool  RowState(
   const int    index,       // index
   const bool   select       // state
   )
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Return Value

true - successful, otherwise - false.

Select

Selects the current item.

```
void  Select(
   const int   index       // index
   )
```

Parameters

index

[in] Item index to select.

Return Value

None.

CSpinEdit

CSpinEdit is a class of SpinEdit complex control (with dependent controls).

Description

CSpinEdit class is intended for creation of a control, which allows editing a value of integer type with a specified step and within specified limitations.

Declaration

```
class CSpinEdit : public CWndContainer
```

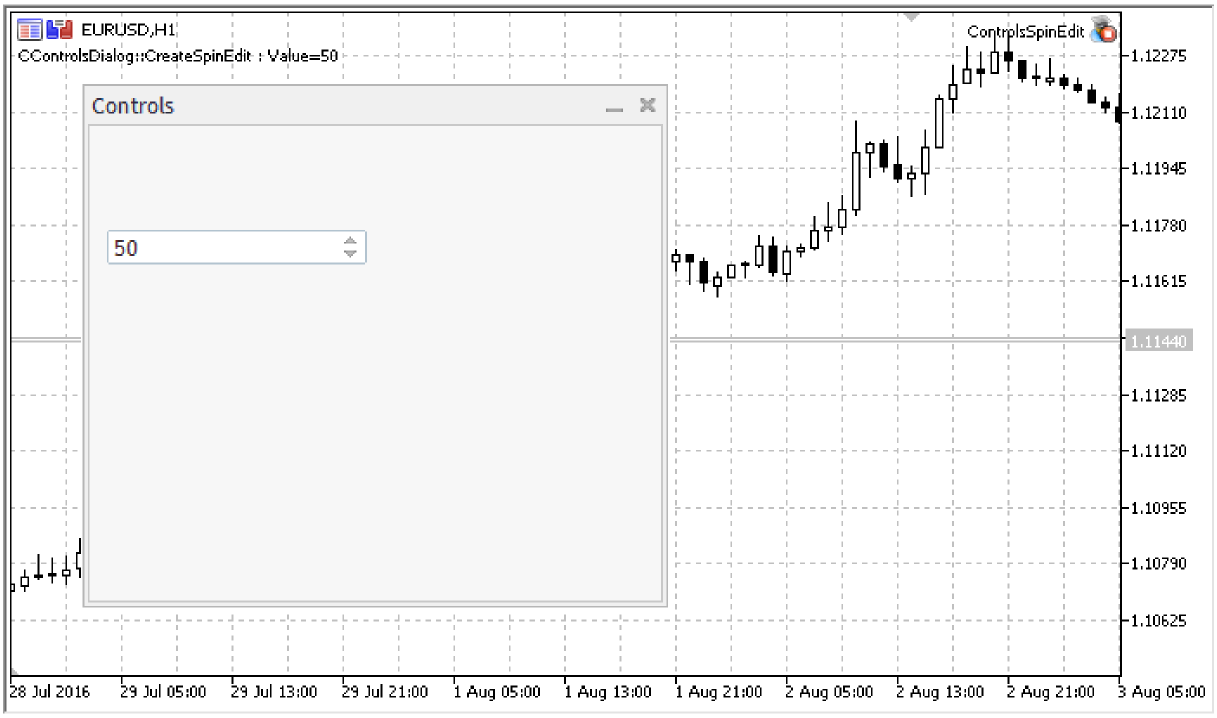
Title

```
#include <Controls\SpinEdit.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CSpinEdit

Result of the code provided below:



CControlsDialog::CreateSpinEdit : Value=50

EURUSD,H1
Controls
50
ControlsSpinEdit
1.12275
1.12110
1.11945
1.11780
1.11615
1.11440
1.11285
1.11120
1.10955
1.10790
1.10625
28 Jul 2016
29 Jul 05:00
29 Jul 13:00
29 Jul 21:00
1 Aug 05:00
1 Aug 13:00
1 Aug 21:00
2 Aug 05:00
2 Aug 13:00
2 Aug 21:00
3 Aug 05:00

Class Methods by Groups

Create	
Create	Creates control

Create	
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
MinValue	Gets/sets the minimal allowed value
MaxValue	Gets/sets the maximal allowed value
State	
Value	Gets/sets the current value
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateInc	Creates dependent control (increment button)
CreateDec	Creates dependent control (decrement button)
Dependent controls event handlers	
OnClickInc	"ClickInc" internal event handler (virtual)
OnClickDec	"ClickDec" internal event handler (virtual)
Internal event handlers	
OnChangeValue	"ChangeValue" internal event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

Example of creating a panel with spin edit control:

```
//+------------------------------------------------------------------+
//|                                             ControlsSpinEdit.mq5 |
//|                        Copyright 2000-2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CSpinEdit"
#include <Controls\Dialog.mqh>
#include <Controls\SpinEdit.mqh>
//+------------------------------------------------------------------+
//| defines                                                          |
//+------------------------------------------------------------------+
//--- indents and gaps
#define INDENT_LEFT                         (11)      // indent from left (with allowa
#define INDENT_TOP                          (11)      // indent from top (with allowar
#define INDENT_RIGHT                        (11)      // indent from right (with allow
#define INDENT_BOTTOM                       (11)      // indent from bottom (with allo
#define CONTROLS_GAP_X                      (5)       // gap by X coordinate
#define CONTROLS_GAP_Y                      (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                        (100)     // size by X coordinate
#define BUTTON_HEIGHT                       (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                         (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                         (150)     // size by X coordinate
#define LIST_HEIGHT                         (179)     // size by Y coordinate
#define RADIO_HEIGHT                        (56)      // size by Y coordinate
#define CHECK_HEIGHT                        (93)      // size by Y coordinate
//+------------------------------------------------------------------+
//| Class CControlsDialog                                            |
//| Usage: main dialog of the Controls application                   |
//+------------------------------------------------------------------+
class CControlsDialog : public CAppDialog
  {
private:
   CSpinEdit         m_spin_edit;                     // CSpinEdit object

public:
                     CControlsDialog(void);
                    ~CControlsDialog(void);
   //--- create
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- chart event handler
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,cons

protected:
   //--- create dependent controls
   bool              CreateSpinEdit(void);
   //--- handlers of the dependent controls events
   void              OnChangeSpinEdit(void);
  };
```

```
//+------------------------------------------------------------------+
//| Event Handling                                                   |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CControlsDialog)
   ON_EVENT(ON_CHANGE,m_spin_edit,OnChangeSpinEdit)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CControlsDialog::CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CControlsDialog::~CControlsDialog(void)
  {
  }
//+------------------------------------------------------------------+
//| Create                                                           |
//+------------------------------------------------------------------+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
  {
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- create dependent controls
   if(!CreateSpinEdit())
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the "SpinEdit" element                                    |
//+------------------------------------------------------------------+
bool CControlsDialog::CreateSpinEdit(void)
  {
//--- coordinates
   int x1=INDENT_LEFT;
   int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+(BUTTON_HEIGHT+CONTROLS_GAP_Y);
   int x2=x1+GROUP_WIDTH;
   int y2=y1+EDIT_HEIGHT;
//--- create
   if(!m_spin_edit.Create(m_chart_id,m_name+"SpinEdit",m_subwin,x1,y1,x2,y2))
      return(false);
   if(!Add(m_spin_edit))
      return(false);
   m_spin_edit.MinValue(10);
   m_spin_edit.MaxValue(100);
   m_spin_edit.Value(50);
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Event handler                                                    |
//+------------------------------------------------------------------+
void CControlsDialog::OnChangeSpinEdit(void)
  {
   Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
  }
//+------------------------------------------------------------------+
//| Global Variables                                                 |
//+------------------------------------------------------------------+
CControlsDialog ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create application dialog
   if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
      return(INIT_FAILED);
//--- run application
   ExtDialog.Run();
//--- succeed
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- clear comments
   Comment("");
//--- destroy dialog
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

Create

Creates new CSpinEdit control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,          // ID
   const long&      lparam,      // parameter
   const double&    dparam,      // parameter
   const string&    sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

MinValue (Get method)

Gets the value of "MinValue" property (minimal value) of the control.

```
int  MinValue()  const
```

Return Value

The value of "MinValue" property.

MinValue (Set method)

Sets the value of "MinValue" property (minimal value) of the control.

```
void  MinValue(
   const int  value      // value
   )
```

Parameters

value

[in] New value of "MinValue" property.

Return Value

None.

MaxValue (Get method)

Gets the value of "MaxValue" property (maximal value) of the control.

```
int  MaxValue()  const
```

Return Value

The value of "MaxValue" property.

MaxValue (Set method)

Sets the value of "MaxValue" property (maximal value) of the control.

```
void  MaxValue(
   const int  value       // value
   )
```

Parameters

value

[in] New value of "MaxValue" property.

Return Value

None.

Value (Get method)

Gets the "Value" property (current value) of the control.

```
int  Value()  const
```

Return Value

The "Value" property.

Value (Set method)

Sets the "Value" property (current value) of the control.

```
void  Value(
   const int  value      // value
   )
```

Parameters

value

[in] New "Value" property.

Return Value

None.

CreateEdit

Creates dependent control (CEdit).

```
virtual bool  CreateEdit()
```

Return Value

true - successful, otherwise - false.

CreateInc

Creates dependent control (increment button).

```
virtual bool  CreateInc()
```

Return Value

true - successful, otherwise - false.

CreateDec

Creates dependent control (decrement button).

```
virtual bool  CreateDec()
```

Return Value

true - successful, otherwise - false.

OnClickInc

The virtual handler of the control "ClickInc" (mouse click on the increment button) internal event.

```
virtual bool  OnClickInc()
```

Return Value

true - event processed, otherwise - false.

OnClickDec

The virtual handler of the control "ClickDec" (mouse click on the decrement button) internal event.

```
virtual bool  OnClickDec()
```

Return Value

true - event processed, otherwise - false.

OnChangeValue

The virtual handler of the control "ChangeValue" (changing the current value) internal event.

```
virtual bool  OnChangeValue()
```

Return Value

true - event processed, otherwise - false.

CDialog

CDialog is class of the Dialog complex control.

Description

CDialog class is intended to combine the controls with different functions in the group.

Declaration

```
class CDialog : public CWndContainer
```

Title

```
#include <Controls\Dialog.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CDialog

Direct descendants

- CAppDialog

Class Methods by Groups

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Caption	Gets/sets the value of the "Caption" property
Add	
Add	Adds control to the client area
Dependent controls	
CreateWhiteBorder	Creates dependent control (white border)
CreateBackground	Creates dependent control (background)
CreateCaption	Creates dependent control (caption)
CreateButtonClose	Creates dependent control (close button)
CreateClientArea	Creates dependent control (client area)

Create	
Dependent controls event handlers	
OnClickCaption	"ClickCaption" internal event handler
OnClickButtonClose	"ClickButtonClose" internal event handler
Access to client area properties	
ClientAreaVisible	Sets a value indicating whether the client area is visible
ClientAreaLeft	Gets X coordinate of the upper-left corner of the control client area
ClientAreaTop	Gets Y coordinate of the upper-left corner of the control client area
ClientAreaRight	Gets X coordinate of the lower-right corner of the control client area
ClientAreaBottom	Gets Y coordinate of the lower-right corner of the control client area
ClientAreaWidth	Gets the client area width
ClientAreaHeight	Gets the client area height
Drag event handlers	
OnDialogDragStart	"DialogDragStart" event handler (virtual)
OnDialogDragProcess	"DialogDragProcess" event handler (virtual)
OnDialogDragEnd	"DialogDragEnd" event handler (virtual)

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

Destroy, OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

Create

Creates new CDialog control.

```
virtual bool  Create(
   const long     chart,        // chart ID
   const string   name,         // name
   const int      subwin,       // chart subwindow
   const int      x1,           // coordinate
   const int      y1,           // coordinate
   const int      x2,           // coordinate
   const int      y2            // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

Caption (Get method)

Gets the "Caption" property of the CDialog control.

```
string  MinValue()  const
```

Return Value

The "Caption" property.

Caption (Set method)

Sets the "Caption" property of the CDialog control.

```
bool  Caption(
   const string  text      // text
   )
```

Parameters

text

[in] New value of "Caption" property.

Return Value

true - successful, otherwise - false.

Add

Adds control to the client area by pointer.

```
bool  Add(
   CWnd    *control,         // pointer
   )
```

Parameters

control

[in] Pointer to control.

Return Value

true - successful, otherwise - false.

Add

Adds control to the client area by reference.

```
bool  Add(
   CWnd    &control,         // reference
   )
```

Parameters

control

[in] Reference to control.

Return Value

true - successful, otherwise - false.

CreateWhiteBorder

Creates dependent control (white border).

```
virtual bool  CreateWhiteBorder()
```

Return Value

true - successful, otherwise - false.

CreateBackground

Creates dependent control (background).

```
virtual bool  CreateBackground()
```

Return Value

true - successful, otherwise - false.

CreateCaption

Creates dependent control (caption).

```
virtual bool  CreateCaption()
```

Return Value

true - successful, otherwise - false.

CreateButtonClose

Creates dependent control (close button)

```
virtual bool  CreateButtonClose()
```

Return Value

true - successful, otherwise - false.

CreateClientArea

Creates dependent control (client area).

```
virtual bool  CreateClientArea()
```

Return Value

true - successful, otherwise - false.

OnClickCaption

The virtual handler of the control "ClickCaption" internal event.

```
virtual bool  OnClickCaption()
```

Return Value

true - successful, otherwise - false.

OnClickButtonClose

The virtual handler of the control "ClickButtonClose" internal event.

```
virtual bool  OnClickButtonClose()
```

Return Value

true - successful, otherwise - false.

ClientAreaVisible

Sets a flag indicating whether the client area is visible.

```
bool  ClientAreaVisible(
   const bool  visible      // visibility flag
   )
```

Parameters

visible

[in] Visibility flag.

Return Value

true - successful, otherwise - false.

ClientAreaLeft

Gets X coordinate of the upper-left corner of the control client area.

```
int  ClientAreaLeft()
```

Return Value

The X coordinate of the upper-left corner of the control client area.

ClientAreaTop

Gets Y coordinate of the upper-left corner of the control client area.

```
int  ClientAreaTop()
```

Return Value

The Y coordinate of the upper-left corner of the control client area.

ClientAreaRight

Gets X coordinate of the lower-right corner of the control client area.

```
int  ClientAreaTop()
```

Return Value

The X coordinate of the lower-right corner of the control client area.

ClientAreaBottom

Gets Y coordinate of the lower-right corner of the control client area.

```
int  ClientAreaBottom()
```

Return Value

The Y coordinate of the lower-right corner of the control client area.

ClientAreaWidth

Gets the width of the control client area.

```
int  ClientAreaWidth()
```

Return Value

The width of the client area.

ClientAreaHeight

Gets the height of the control client area.

```
int  ClientAreaHeight()
```

Return Value

The height of the control client area.

OnDialogDragStart

The virtual handler of the control "DialogDragStart" event.

```
virtual bool  OnDialogDragStart()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragStart" event occurs at start of the drag of the control.

OnDialogDragProcess

The virtual handler of the control "DialogDragProcess" event.

```
virtual bool  OnDialogDragProcess()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragProcess" event occurs when the control is dragged.

OnDialogDragEnd

The virtual handler of the control "DialogDragEnd" event.

```
virtual bool  OnDialogDragEnd()
```

Return Value

true - event processed, otherwise - false.

Note

The "DialogDragEnd" event occurs at the end of the drag of the control.

CAppDialog

CAppDialog is a class of Application Dialog complex control (with dependent controls).

Description

CAppDialog class is intended to combine the controls with different functions in the group inside the MQL5 program.

Declaration

```
class CAppDialog : public CDialog
```

Title

```
#include <Controls\Dialog.mqh>
```

Inheritance hierarchy

- CObject
  - CWnd
    - CWndContainer
      - CDialog
        - CAppDialog

Class Methods by Groups

Create and destroy	
Create	Creates control
Destroy	Destroys control
Events processing	
OnEvent	Event handler of all chart events
Run	
Run	Runs control
Chart events processing	
ChartEvent	Event handler of all chart events
Settings	
Minimized	Sets a value indicating whether the control is minimized
Save/Load	
IniFileSave	Saves the control state to file
IniFileLoad	Loads the control state from file
IniFileName	Sets the file name for loading/saving the control state

Create and destroy	
IniFileExt	Sets the file extension for loading/saving the control state
Initialization	
CreateCommon	Common initialization method
CreateExpert	Initialization method for working in Expert Advisors
CreateIndicator	Initialization method for working in indicators
Dependent controls	
CreateButtonMinMax	Creates dependent controls (minimize/maximize buttons)
Dependent controls event handlers	
OnClickButtonClose	"ClickButtonClose" internal event handler (virtual)
OnClickButtonMinMax	"ClickButtonMinMax" internal event handler (virtual)
External events	
OnAnotherApplicationClose	Event handler of external events (virtual)
Methods	
Rebound	Sets new coordinates of the control using CRect class coordinates
Minimize	Shows the control in the minimized state
Maximize	Shows the control in the maximized (restored) state
CreateInstanceId	Creates a unique Id for the names of the control objects
ProgramName	Gets the name of MQL5 program, at which the control is used
SubwinOff	Get the Y offset of the control subwindow

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Methods inherited from class CWnd

Name, ControlsTotal, Control, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Contains, Contains, Alignment, Align, Id, IsEnabled, IsVisible, Visible, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

Methods inherited from class CWndContainer

OnMouseEvent, ControlsTotal, Control, ControlFind, MouseFocusKill, Add, Add, Delete, Delete, Move, Move, Shift, Id, Enable, Disable, Show, Hide

Methods inherited from class CDialog

Methods inherited from class CObject

Prev, Prev, Next, Next, Type, Compare

Caption, Caption, Add, Add

Create

Creates new CAppDialog control.

```
virtual bool  Create(
   const long    chart,       // chart ID
   const string  name,        // name
   const int     subwin,      // chart subwindow
   const int     x1,          // coordinate
   const int     y1,          // coordinate
   const int     x2,          // coordinate
   const int     y2           // coordinate
   )
```

Parameters

chart

[in] ID of the chart, at which the ontrol is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the ontrol is created.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

Destroy

CAppDialog control deinitialization method.

```
virtual void  Destroy(
   const int  reason=REASON_PROGRAM     // reason code
)
```

Parameters

reason

[in] Deinitialization reason code. REASON_PROGRAM is set by default.

Return Value

None.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int        id,           // ID
   const long&      lparam,       // parameter
   const double&    dparam,       // parameter
   const string&    sparam        // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

Run

Runs control.

```
bool  Run()
```

Return Value

true - successful, otherwise - false.

ChartEvent

Virtual handler of the control events.

```
virtual bool  ChartEvent(
   const int       id,          // ID
   const long&     lparam,      // parameter
   const double&   dparam,      // parameter
   const string&   sparam       // parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type passed by reference.

dparam

[in] Event parameter of double type passed by reference.

sparam

[in] Event parameter of string type passed by reference.

Return Value

true - event processed, otherwise - false.

Minimized

Sets the value of "Minimized" (window state) property of the control.

```
bool  Minimized(
   const bool  flag      // state
   )
```

Parameters

flag

[in] New state.

Return Value

true - successful, otherwise - false.

IniFileSave

Saves the control state to file.

```
void  IniFileSave()
```

Return Value

true - successful, otherwise - false.

IniFileLoad

Loads the control state from file.

```
void  IniFileLoad()
```

Return Value

true - successful, otherwise - false.

IniFileName

Sets the file name for loading/saving the control state.

```
virtual string  IniFileName()  const
```

Return Value

File name for loading/saving of the control state.

Note

The file name includes the name of the Expert Advisor/indicator and working symbol, on which MQL5 program is launched.

IniFileExt

Sets the file extension for loading/saving the control state.

```
virtual string  IniFileExt()  const
```

Return Value

File extension used for loading/saving of the control state.

CreateCommon

Common initialization method.

```
bool  CreateCommon(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   )
```

Parameters

chart

[in] ID of the chart, at which the control is created.

name

[in] Unique name of the control.

subwin

[in] Subwindow of the chart, at which the control is created.

Return Value

true - successful, otherwise - false.

CreateExpert

Initialization method for working in Expert Advisors.

```
bool  CreateExpert(
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

CreateIndicator

Initialization method for working in indicators.

```
bool  CreateIndicator(
   const int      x1,          // coordinate
   const int      y1,          // coordinate
   const int      x2,          // coordinate
   const int      y2           // coordinate
   )
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Return Value

true - successful, otherwise - false.

CreateButtonMinMax

Creates dependent controls (minimize/maximize buttons).

```
virtual void  CreateButtonMinMax()
```

Return Value

None.

OnClickButtonClose

The virtual handler of the control "ClickButtonClose" (mouse click on close button) internal event.

```
virtual void  OnClickButtonClose()
```

Return Value

None.

OnClickButtonMinMax

The virtual handler of the control "ClickButtonMinMax" (mouse click on minimize/maximize button) internal event.

```
virtual void  OnClickButtonClose()
```

Return Value

None.

OnAnotherApplicationClose

The virtual handler of the control external events.

```
virtual void  OnAnotherApplicationClose()
```

Return Value

None.

Rebound

Sets new coordinates of the control using CRect class coordinates.

```
bool  Rebound(
   const & CRect  rect      // CRect class
   )
```

Return Value

true - successful, otherwise - false.

Minimize

Shows the control window in the minimized state.

```
virtual void  Minimize()
```

Return Value

true - successful, otherwise - false.

Maximize

Shows the control window in the maximized (restored) state.

```
virtual void  Maximize()
```

Return Value

true - successful, otherwise - false.

CreateInstanceId

Creates a unique prefix for the names of the control objects.

```
string  CreateInstanceId()
```

Return Value

Prefix for the object names.

ProgramName

Gets the name of the MQL5 program, at which the control is used.

```
string  ProgramName()
```

Return Value

Name of the MQL5 program.

SubwinOff

Gets the Y offset of the control subwindow.

```
void  SubwinOff()
```

Return Value

None.

Moving from MQL4 to MQL5

MQL5 is the evolution of its predecessor - the MQL4 programming language, in which numerous indicators, scripts, and Expert Advisors were written. Despite the fact that the new programming language is maximally compatible with the previous-generation language, there are still some differences between these languages. And when transferring programs these differences should be noted.

This section contains information intended to facilitate the adaptation of codes to the new MQL5 language for programmers who know MQL4.

First it should be noted:

- The new language does not contain functions start(), init() and deinit().
- No limit for the number of indicator buffers.
- DLLs are loaded immediately after loading an Expert Advisor (or any other mql5 program).
- Check of logical conditions is shortened.
- When limits of an array are exceeded, the current performance is terminated (critically - with the output of an errors).
- Precedence of operators like in C + +.
- The language offers the implicit type cast (even from string to a number).
- Local variables are not initialized automatically (except for strings).
- Common local arrays are automatically deleted.

Special Functions init, start and deinit

The MQL4 language contained only three predefined functions that could be used in the indicator, script or Expert Advisor (not taking into account the include files *.mqh and library files). In MQL5 there are no such functions, but there are their analogues. The table shows the approximate correspondence of functions.

MQL4	MQL5
init	OnInit
start	OnStart
deinit	OnDeinit

Functions OnInit and OnDeinit perform the same role as init and deinit in MQL4 - they are designed to locate the code, which must be performed during initialization and deinitialization of mql5 programs. You can either just rename these functions accordingly, or leave them as they are, but add calls of these functions in corresponding places.

Example:

```
void OnInit()
  {
//--- Call function upon initialization
   init();
  }
void OnDeinit(const int reason)
  {
//--- Call function upon deinitialization
   deinit();
//---
  }
```

The start function is replaced by OnStart only in scripts. In Expert Advisors and indicators it should be renamed to OnTick and OnCalculate, respectively. The code that is to be executed during a mql5 program operation should be located in these three functions:

mql5-program	main function
script	OnStart
indicator	OnCalculate
Expert Advisor	OnTick

If the indicator or script code does not contain the main function, or the function name differs from the required one, the call of this function is not performed. It means, if the source code of a script doesn't contain OnStart, such a code will be compiled as an Expert Advisor.

If an indicator code doesn't contain the OnCalculate function, the compilation of such an indicator is impossible.

Predefined Variables

In MQL5 there are no such predefined variables as Ask, Bid, Bars. Variables Point and Digits have a slightly different spelling:

MQL4	MQL5
Digits	_Digits
Point	_Point
	_LastError
	_Period
	_Symbol
	_StopFlag
	_UninitReason

Access to Timeseries

In MQL5 there are no such predefined timeseries as Open[], High[], Low[], Close[], Volume[] and Time[]. The necessary depth of a timeseries can now be set using corresponding functions to access timeseries.

Expert Advisors

Expert Advisors in MQL5 do not require the obligatory presence of functions that handle the events of a new tick receipt - OnTick, as it was in MQL4 (the start function in MQL4 is executed when a new tick is received). In MQL5 Expert Advisors can contain pre-defined handler functions of several types of events:

- OnTick – receipt of a new tick;
- OnTimer – timer event;
- OnTrade - trade event;
- OnChartEvent – events of input from the keyboard and mouse, events of a graphic object moving, event of a text editing completion in the entry field of the LabelEdit object;
- OnBookEvent – event of Depth of Market status change.

Custom Indicators

In MQL4, the number of indicator buffers is limited and can't exceed 8. In MQL5 there are no such limitations, but it should be remembered that each indicator buffer requires allocation of a certain part of memory for its location in the terminal, so the new possibility should not be abused.

MQL4 offered only 6 types of custom indicator plotting; while MQL5 now offers 18 drawing styles. The names of drawing types haven't changed, but the ideology of the graphical representation of indicators has changed significantly.

The direction of indexing in indicator buffers also differs. By default, in MQL5 all the indicator buffers have the behavior of common arrays, i.e. 0 indexed element is the oldest one in the history, and as the index increases, we move from the oldest data to the most recent ones.

The only function for working with custom indicators that was preserved from MQL4 is SetIndexBuffer. But its call has changed; now you should specify type of data to be stored in an array, linked to the indicator buffer.

Properties of custom indicators also have changed and expanded. New functions for accessing timeseries have been added, so the total calculation algorithm must be reconsidered.

Graphical Objects

The number of graphical objects in MQL5 has been significantly increased. Besides, graphical objects can now be positioned in time with the accuracy of a second in a chart of any timeframe - now object anchor points are not rounded off to the bar opening time in the current price chart.

For the Arrow, Text and Label objects now you can specify binding methods, and for the Label, Button, Chart, Bitmap Label and Edit objects you can set chart corner to which an object is attached.

List of MQL5 Functions

All MQL5 functions in alphabetical order.

Function	Action	Section
AccountInfoDouble	Returns a value of double type of the corresponding account property	Account Information
AccountInfoInteger	Returns a value of integer type (bool, int or long) of the corresponding account property	Account Information
AccountInfoString	Returns a value string type corresponding account property	Account Information
acos	Returns the arc cosine of x in radians	Math Functions
Alert	Displays a message in a separate window	Common Functions
ArrayBsearch	Searches for a specified value in a multidimensional numeric array sorted ascending	Array Functions
ArrayCompare	Returns the result of comparing two arrays of simple types or custom structures without complex objects	Array Functions
ArrayCopy	Copies one array into another	Array Functions
ArrayFill	Fills an array with the specified value	Array Functions
ArrayFree	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.	Array Functions
ArrayGetAsSeries	Checks direction of array indexing	Array Functions
ArrayInitialize	Sets all elements of a numeric array into a single value	Array Functions
ArrayIsDynamic	Checks whether an array is dynamic	Array Functions
ArrayIsSeries	Checks whether an array is a timeseries	Array Functions
ArrayMaximum	Searches for the largest element in the first dimension of a multidimensional numeric array	Array Functions

Function	Action	Section
ArrayMinimum	Searches for the lowest element in the first dimension of a multidimensional numeric array	Array Functions
ArrayRange	Returns the number of elements in the specified dimension of the array	Array Functions
ArrayResize	Sets the new size in the first dimension of the array	Array Functions
ArraySetAsSeries	Sets the direction of array indexing	Array Functions
ArraySize	Returns the number of elements in the array	Array Functions
ArraySort	Sorting of numeric arrays by the first dimension	Array Functions
ArrayPrint	Prints an array of a simple type or a simple structure into journal	Array Functions
ArrayInsert	Inserts the specified number of elements from a source array to a receiving one starting from a specified index	Array Functions
ArrayRemove	Removes the specified number of elements from the array starting with a specified index	Array Functions
ArrayReverse	Reverses the specified number of elements in the array starting with a specified index	Array Functions
ArraySwap	Swaps the contents of two dynamic arrays of the same type	Array Functions
asin	Returns the arc sine of x in radians	Math Functions
atan	Returns the arc tangent of x in radians	Math Functions
Bars	Returns the number of bars the history for a specified symbol and period	Timeseries and Indicators Access
BarsCalculated	Returns the number of calculated data in an indicator buffer or -1 in the case of error (data hasn't been calculated yet)	Timeseries and Indicators Access

Function	Action	Section
CalendarCountryById	Get a country description by its ID	Economic Calendar
CalendarEventById	Get an event description by its ID	Economic Calendar
CalendarValueById	Get an event value description by its ID	Economic Calendar
CalendarCountries	Get the array of country names available in the calendar	Economic Calendar
CalendarEventByCountry	Get the array of descriptions of all events available in the calendar by a specified country code	Economic Calendar
CalendarEventByCurrency	Get the array of descriptions of all events available in the calendar by a specified currency	Economic Calendar
CalendarValueHistoryByEvent	Get the array of values for all events in a specified time range by an event ID	Economic Calendar
CalendarValueHistory	Get the array of values for all events in a specified time range with the ability to sort by country and/or currency	Economic Calendar
CalendarValueLastByEvent	Get the array of event values by its ID since the calendar database status with a specified change_id	Economic Calendar
CalendarValueLast	Get the array of values for all events with the ability to sort by country and/or currency since the calendar database status with a specified change_id	Economic Calendar
ceil	Returns integer numeric value closest from above	Math Functions
CharArrayToString	Converting symbol code (ansi) into one-symbol array	Conversion Functions
ChartApplyTemplate	Applies a specific template from a specified file to the chart	Chart Operations
ChartClose	Closes the specified chart	Chart Operations
ChartFirst	Returns the ID of the first chart of the client terminal	Chart Operations

Function	Action	Section
ChartGetDouble	Returns the double value property of the specified chart	Chart Operations
ChartGetInteger	Returns the integer value property of the specified chart	Chart Operations
ChartGetString	Returns the string value property of the specified chart	Chart Operations
ChartID	Returns the ID of the current chart	Chart Operations
ChartIndicatorAdd	Adds an indicator with the specified handle into a specified chart window	Chart Operations
ChartIndicatorDelete	Removes an indicator with a specified name from the specified chart window	Chart Operations
ChartIndicatorGet	Returns the handle of the indicator with the specified short name in the specified chart window	Chart Operations
ChartIndicatorName	Returns the short name of the indicator by the number in the indicators list on the specified chart window	Chart Operations
ChartIndicatorsTotal	Returns the number of all indicators applied to the specified chart window.	Chart Operations
ChartNavigate	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart	Chart Operations
ChartNext	Returns the chart ID of the chart next to the specified one	Chart Operations
ChartOpen	Opens a new chart with the specified symbol and period	Chart Operations
CharToString	Converting a symbol code into a one-character string	Conversion Functions
ChartPeriod	Returns the period value of the specified chart	Chart Operations
ChartPriceOnDropped	Returns the price coordinate of the chart point, the Expert	Chart Operations

Function	Action	Section
	Advisor or script has been dropped to	
ChartRedraw	Calls a forced redrawing of a specified chart	Chart Operations
ChartSaveTemplate	Saves current chart settings in a template with a specified name	Chart Operations
ChartScreenShot	Provides a screenshot of the chart of its current state in a GIF, PNG or BMP format depending on specified extension	Chart Operations
ChartSetDouble	Sets the double value for a corresponding property of the specified chart	Chart Operations
ChartSetInteger	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart	Chart Operations
ChartSetString	Sets the string value for a corresponding property of the specified chart	Chart Operations
ChartSetSymbolPeriod	Changes the symbol value and a period of the specified chart	Chart Operations
ChartSymbol	Returns the symbol name of the specified chart	Chart Operations
ChartTimeOnDropped	Returns the time coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
ChartTimePriceToXY	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates	Chart Operations
ChartWindowFind	Returns the number of a subwindow where an indicator is drawn	Chart Operations
ChartWindowOnDropped	Returns the number (index) of the chart subwindow, the Expert Advisor or script has been dropped to	Chart Operations

Function	Action	Section
ChartXOnDropped	Returns the X coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
ChartXYToTimePrice	Converts the X and Y coordinates on a chart to the time and price values	Chart Operations
ChartYOnDropped	Returns the Y coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
CheckPointer	Returns the type of the object pointer	Common Functions
CLBufferCreate	Creates an OpenCL buffer	Working with OpenCL
CLBufferFree	Deletes an OpenCL buffer	Working with OpenCL
CLBufferRead	Reads an OpenCL buffer into an array	Working with OpenCL
CLBufferWrite	Writes an array into an OpenCL buffer	Working with OpenCL
CLContextCreate	Creates an OpenCL context	Working with OpenCL
CLContextFree	Removes an OpenCL context	Working with OpenCL
CLExecute	Runs an OpenCL program	Working with OpenCL
CLGetDeviceInfo	Receives device property from OpenCL driver	Working with OpenCL
CLGetInfoInteger	Returns the value of an integer property for an OpenCL object or device	Working with OpenCL
CLHandleType	Returns the type of an OpenCL handle as a value of the ENUM_OPENCL_HANDLE_TYPE enumeration	Working with OpenCL
CLKernelCreate	Creates an OpenCL start function	Working with OpenCL
CLKernelFree	Removes an OpenCL start function	Working with OpenCL
CLProgramCreate	Creates an OpenCL program from a source code	Working with OpenCL
CLProgramFree	Removes an OpenCL program	Working with OpenCL
CLSetKernelArg	Sets a parameter for the OpenCL function	Working with OpenCL

Function	Action	Section
CLSetKernelArgMem	Sets an OpenCL buffer as a parameter of the OpenCL function	Working with OpenCL
ColorToARGB	Converting color type to uint type to receive ARGB representation of the color.	Conversion Functions
ColorToString	Converting color value into string as "R,G,B"	Conversion Functions
Comment	Outputs a comment in the left top corner of the chart	Common Functions
CopyBuffer	Gets data of a specified buffer from a specified indicator into an array	Timeseries and Indicators Access
CopyClose	Gets history data on bar closing price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyHigh	Gets history data on maximal bar price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyLow	Gets history data on minimal bar price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyOpen	Gets history data on bar opening price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyRates	Gets history data of the Rates structure for a specified symbol and period into an array	Timeseries and Indicators Access
CopyRealVolume	Gets history data on trade volumes for a specified symbol and period into an array	Timeseries and Indicators Access
CopySpread	Gets history data on spreads for a specified symbol and period into an array	Timeseries and Indicators Access
CopyTicks	Gets ticks accumulated by the terminal for the current working session into an array	Timeseries and Indicators Access
CopyTickVolume	Gets history data on tick volumes for a specified symbol and period into an array	Timeseries and Indicators Access

Function	Action	Section
CopyTime	Gets history data on bar opening time for a specified symbol and period into an array	Timeseries and Indicators Access
cos	Returns the cosine of a number	Math Functions
CryptDecode	Performs the inverse transformation of the data from array	Common Functions
CryptEncode	Transforms the data from array with the specified method	Common Functions
CustomSymbolCreate	Create a custom symbol with the specified name in the specified group	Custom Symbols
CustomSymbolDelete	Delete a custom symbol with the specified name	Custom Symbols
CustomSymbolSetInteger	Set the integer type property value for a custom symbol	Custom Symbols
CustomSymbolSetDouble	Set the real type property value for a custom symbol	Custom Symbols
CustomSymbolSetString	Set the string type property value for a custom symbol	Custom Symbols
CustomSymbolSetMarginRate	Set the margin rates depending on the order type and direction for a custom symbol	Custom Symbols
CustomSymbolSetSessionQuote	Set the start and end time of the specified quotation session for the specified symbol and week day	Custom Symbols
CustomSymbolSetSessionTrade	Set the start and end time of the specified trading session for the specified symbol and week day	Custom Symbols
CustomRatesDelete	Delete all bars from the price history of the custom symbol in the specified time interval	Custom Symbols
CustomRatesReplace	Fully replace the price history of the custom symbol within the specified time interval with the data from the MqlRates type array	Custom Symbols

Function	Action	Section
CustomRatesUpdate	Add missing bars to the custom symbol history and replace existing data with the ones from the MqlRates type array	Custom Symbols
CustomTicksAdd	Adds data from an array of the MqlTick type to the price history of a custom symbol. The custom symbol must be selected in the Market Watch window	Custom Symbols
CustomTicksDelete	Delete all ticks from the price history of the custom symbol in the specified time interval	Custom Symbols
CustomTicksReplace	Fully replace the price history of the custom symbol within the specified time interval with the data from the MqlTick type array	Custom Symbols
CustomBookAdd	Passes the status of the Depth of Market for a custom symbol	Custom Symbols
DatabaseOpen	Opens or creates a database in a specified file	Working with databases
DatabaseClose	Closes a database	Working with databases
DatabaseImport	Imports data from a file into a table	Working with databases
DatabaseExport	Exports a table or an SQL request execution result to a CSV file	Working with databases
DatabasePrint	Prints a table or an SQL request execution result in the Experts journal	Working with databases
DatabaseTableExists	Checks the presence of the table in a database	Working with databases
DatabaseExecute	Executes a request to a specified database	Working with databases
DatabasePrepare	Creates a handle of a request, which can then be executed using DatabaseRead()	Working with databases
DatabaseReset	Resets a request, like after calling DatabasePrepare()	Working with databases

Function	Action	Section
DatabaseBind	Sets a parameter value in a request	Working with databases
DatabaseBindArray	Sets an array as a parameter value	Working with databases
DatabaseRead	Moves to the next entry as a result of a request	Working with databases
DatabaseReadBind	Moves to the next record and reads data into the structure from it	Working with databases
DatabaseFinalize	Removes a request created in DatabasePrepare()	Working with databases
DatabaseTransactionBegin	Starts transaction execution	Working with databases
DatabaseTransactionCommit	Completes transaction execution	Working with databases
DatabaseTransactionRollback	Rolls back transactions	Working with databases
DatabaseColumnsCount	Gets the number of fields in a request	Working with databases
DatabaseColumnName	Gets a field name by index	Working with databases
DatabaseColumnType	Gets a field type by index	Working with databases
DatabaseColumnSize	Gets a field size in bytes	Working with databases
DatabaseColumnText	Gets a field value as a string from the current record	Working with databases
DatabaseColumnInteger	Gets the int type value from the current record	Working with databases
DatabaseColumnLong	Gets the long type value from the current record	Working with databases
DatabaseColumnDouble	Gets the double type value from the current record	Working with databases
DatabaseColumnBlob	Gets a field value as an array from the current record	Working with databases
DebugBreak	Program breakpoint in debugging	Common Functions
Digits	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol	Checkup
DoubleToString	Converting a numeric value to a text line with a specified	Conversion Functions

Function	Action	Section
	accuracy	
DXContextCreate	Creates a graphic context for rendering frames of a specified size	Working with DirectX
DXContextSetSize	Changes a frame size of a graphic context created in DXContextCreate()	Working with DirectX
DXContextSetSize	Gets a frame size of a graphic context created in DXContextCreate()	Working with DirectX
DXContextClearColors	Sets a specified color to all pixels for the rendering buffer	Working with DirectX
DXContextClearDepth	Clears the depth buffer	Working with DirectX
DXContextGetColors	Gets an image of a specified size and offset from a graphic context	Working with DirectX
DXContextGetDepth	Gets the depth buffer of a rendered frame	Working with DirectX
DXBufferCreate	Creates a buffer of a specified type based on a data array	Working with DirectX
DXTextureCreate	Creates a 2D texture out of a rectangle of a specified size cut from a passed image	Working with DirectX
DXInputCreate	Creates shader inputs	Working with DirectX
DXInputSet	Sets shader inputs	Working with DirectX
DXShaderCreate	Creates a shader of a specified type	Working with DirectX
DXShaderSetLayout	Sets vertex layout for the vertex shader	Working with DirectX
DXShaderInputsSet	Sets shader inputs	Working with DirectX
DXShaderTexturesSet	Sets shader textures	Working with DirectX
DXDraw	Renders the vertices of the vertex buffer set in DXBufferSet()	Working with DirectX
DXDrawIndexed	Renders graphic primitives described by the index buffer from DXBufferSet()	Working with DirectX

Function	Action	Section
DXPrimiveTopologySet	Sets the type of primitives for rendering using DXDrawIndexed()	Working with DirectX
DXBufferSet	Sets a buffer for the current rendering	Working with DirectX
DXShaderSet	Sets a shader for rendering	Working with DirectX
DXHandleType	Returns a handle type	Working with DirectX
DXRelease	Releases a handle	Working with DirectX
EnumToString	Converting an enumeration value of any type to string	Conversion Functions
EventChartCustom	Generates a custom event for the specified chart	Working with Events
EventKillTimer	Stops the generation of events by the timer in the current chart	Working with Events
EventSetMillisecondTimer	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart	Working with Events
EventSetTimer	Starts the timer event generator with the specified periodicity for the current chart	Working with Events
exp	Returns exponent of a number	Math Functions
ExpertRemove	Stops Expert Advisor and unloads it from the chart	Common Functions
fabs	Returns absolute value (modulus) of the specified numeric value	Math Functions
FileClose	Closes a previously opened file	File Functions
FileCopy	Copies the original file from a local or shared folder to another file	File Functions
FileDelete	Deletes a specified file	File Functions
FileFindClose	Closes search handle	File Functions
FileFindFirst	Starts the search of files in a directory in accordance with the specified filter	File Functions
FileFindNext	Continues the search started by the FileFindFirst() function	File Functions

Function	Action	Section
FileFlush	Writes to a disk all data remaining in the input/output file buffer	File Functions
FileGetInteger	Gets an integer property of a file	File Functions
FileIsEnding	Defines the end of a file in the process of reading	File Functions
FileIsExist	Checks the existence of a file	File Functions
FileIsLineEnding	Defines the end of a line in a text file in the process of reading	File Functions
FileMove	Moves or renames a file	File Functions
FileOpen	Opens a file with a specified name and flag	File Functions
FileReadArray	Reads arrays of any type except for string from the file of the BIN type	File Functions
FileReadBool	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type	File Functions
FileReadDatetime	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value	File Functions
FileReadDouble	Reads a double value from the current position of the file pointer	File Functions
FileReadFloat	Reads a float value from the current position of the file pointer	File Functions
FileReadInteger	Reads int, short or char value from the current position of the file pointer	File Functions
FileReadLong	Reads a long type value from the current position of the file pointer	File Functions

Function	Action	Section
FileReadNumber	Reads from the file of the CSV type a string from the current position till a delimiter (or til the end of a text line) and converts the read string into double value	File Functions
FileReadString	Reads a string from the current position of a file pointer from a file	File Functions
FileReadStruct	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer	File Functions
FileSeek	Moves the position of the file pointer by a specified number of bytes relative to the specified position	File Functions
FileSize	Returns the size of a corresponding open file	File Functions
FileTell	Returns the current position of the file pointer of a corresponding open file	File Functions
FileWrite	Writes data to a file of CSV or TXT type	File Functions
FileWriteArray	Writes arrays of any type except for string into a file of BIN type	File Functions
FileWriteDouble	Writes value of the double type from the current position of a file pointer into a binary file	File Functions
FileWriteFloat	Writes value of the float type from the current position of a file pointer into a binary file	File Functions
FileWriteInteger	Writes value of the int type from the current position of a file pointer into a binary file	File Functions
FileWriteLong	Writes value of the long type from the current position of a file pointer into a binary file	File Functions
FileWriteString	Writes the value of a string parameter into a BIN or TXT file	File Functions

Function	Action	Section
	starting from the current position of the file pointer	
FileWriteStruct	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer	File Functions
floor	Returns integer numeric value closest from below	Math Functions
fmax	Returns the maximal value of the two numeric values	Math Functions
fmin	Returns the minimal value of the two numeric values	Math Functions
fmod	Returns the real remainder after the division of two numbers	Math Functions
FolderClean	Deletes all files in the specified folder	File Functions
FolderCreate	Creates a folder in the Files directory	File Functions
FolderDelete	Removes a selected directory. If the folder is not empty, then it can't be removed	File Functions
FrameAdd	Adds a frame with data	Working with Optimization Results
FrameFilter	Sets the frame reading filter and moves the pointer to the beginning	Working with Optimization Results
FrameFirst	Moves a pointer of frame reading to the beginning and resets the previously set filter	Working with Optimization Results
FrameInputs	Receives input parameters, on which the frame is formed	Working with Optimization Results
FrameNext	Reads a frame and moves the pointer to the next one	Working with Optimization Results
GetLastError	Returns the last error	Checkup
GetPointer	Returns the object pointer	Common Functions
GetTickCount	Returns the number of milliseconds that have elapsed since the system was started	Common Functions

Function	Action	Section
GlobalVariableCheck	Checks the existence of a global variable with the specified name	Global Variables of the Terminal
GlobalVariableDel	Deletes a global variable	Global Variables of the Terminal
GlobalVariableGet	Returns the value of a global variable	Global Variables of the Terminal
GlobalVariableName	Returns the name of a global variable by its ordinal number in the list of global variables	Global Variables of the Terminal
GlobalVariablesDeleteAll	Deletes global variables with the specified prefix in their names	Global Variables of the Terminal
GlobalVariableSet	Sets the new value to a global variable	Global Variables of the Terminal
GlobalVariableSetOnCondition	Sets the new value of the existing global variable by condition	Global Variables of the Terminal
GlobalVariablesFlush	Forcibly saves contents of all global variables to a disk	Global Variables of the Terminal
GlobalVariablesTotal	Returns the total number of global variables	Global Variables of the Terminal
GlobalVariableTemp	Sets the new value to a global variable, that exists only in the current session of the terminal	Global Variables of the Terminal
GlobalVariableTime	Returns time of the last accessing the global variable	Global Variables of the Terminal
HistoryDealGetDouble	Returns the requested property of a deal in the history (double)	Trade Functions
HistoryDealGetInteger	Returns the requested property of a deal in the history (datetime or int)	Trade Functions
HistoryDealGetString	Returns the requested property of a deal in the history (string)	Trade Functions
HistoryDealGetTicket	Returns a ticket of a corresponding deal in the history	Trade Functions
HistoryDealSelect	Selects a deal in the history for further calling it through appropriate functions	Trade Functions
HistoryDealsTotal	Returns the number of deals in the history	Trade Functions

Function	Action	Section
HistoryOrderGetDouble	Returns the requested property of an order in the history (double)	Trade Functions
HistoryOrderGetInteger	Returns the requested property of an order in the history (datetime or int)	Trade Functions
HistoryOrderGetString	Returns the requested property of an order in the history (string)	Trade Functions
HistoryOrderGetTicket	Return order ticket of a corresponding order in the history	Trade Functions
HistoryOrderSelect	Selects an order in the history for further working with it	Trade Functions
HistoryOrdersTotal	Returns the number of orders in the history	Trade Functions
HistorySelect	Retrieves the history of transactions and orders for the specified period of the server time	Trade Functions
HistorySelectByPosition	Requests the history of deals with a specified position identifier.	Trade Functions
iBars	Returns the number of bars of a corresponding symbol and period, available in history	Timeseries and Indicators Access
iBarShift	Returns the index of the bar corresponding to the specified time	Timeseries and Indicators Access
iClose	Returns the Close price of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iHigh	Returns the High price of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iHighest	Returns the index of the highest value found on the corresponding chart (shift relative to the current bar)	Timeseries and Indicators Access

Function	Action	Section
iLow	Returns the Low price of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iLowest	Returns the index of the smallest value found on the corresponding chart (shift relative to the current bar)	Timeseries and Indicators Access
iOpen	Returns the Open price of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iTime	Returns the opening time of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iTickVolume	Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iRealVolume	Returns the real volume of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iVolume	Returns the tick volume of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iSpread	Returns the spread value of the bar (indicated by the 'shift' parameter) on the corresponding chart	Timeseries and Indicators Access
iAD	Accumulation/Distribution	Technical Indicators
iADX	Average Directional Index	Technical Indicators
iADXWilder	Average Directional Index by Welles Wilder	Technical Indicators
iAlligator	Alligator	Technical Indicators
iAMA	Adaptive Moving Average	Technical Indicators
iAO	Awesome Oscillator	Technical Indicators
iATR	Average True Range	Technical Indicators

Function	Action	Section
iBands	Bollinger Bands®	Technical Indicators
iBearsPower	Bears Power	Technical Indicators
iBullsPower	Bulls Power	Technical Indicators
iBWMFI	Market Facilitation Index by Bill Williams	Technical Indicators
iCCI	Commodity Channel Index	Technical Indicators
iChaikin	Chaikin Oscillator	Technical Indicators
iCustom	Custom indicator	Technical Indicators
iDEMA	Double Exponential Moving Average	Technical Indicators
iDeMarker	DeMarker	Technical Indicators
iEnvelopes	Envelopes	Technical Indicators
iForce	Force Index	Technical Indicators
iFractals	Fractals	Technical Indicators
iFrAMA	Fractal Adaptive Moving Average	Technical Indicators
iGator	Gator Oscillator	Technical Indicators
iIchimoku	Ichimoku Kinko Hyo	Technical Indicators
iMA	Moving Average	Technical Indicators
iMACD	Moving Averages Convergence-Divergence	Technical Indicators
iMFI	Money Flow Index	Technical Indicators
iMomentum	Momentum	Technical Indicators
IndicatorCreate	Returns the handle to the specified technical indicator created by an array of MqlParam type parameters	Timeseries and Indicators Access
IndicatorParameters	Based on the specified handle, returns the number of input parameters of the indicator, as well as the values and types of the parameters	Timeseries and Indicators Access
IndicatorRelease	Removes an indicator handle and releases the calculation block of the indicator, if it's not used by anyone else	Timeseries and Indicators Access

Function	Action	Section
IndicatorSetDouble	Sets the value of an indicator property of the double type	Custom Indicators
IndicatorSetInteger	Sets the value of an indicator property of the int type	Custom Indicators
IndicatorSetString	Sets the value of an indicator property of the string type	Custom Indicators
IntegerToString	Converting int into a string of preset length	Conversion Functions
iOBV	On Balance Volume	Technical Indicators
iOsMA	Moving Average of Oscillator (MACD histogram)	Technical Indicators
iRSI	Relative Strength Index	Technical Indicators
iRVI	Relative Vigor Index	Technical Indicators
iSAR	Parabolic Stop And Reverse System	Technical Indicators
IsStopped	Returns true, if an mql5 program has been commanded to stop its operation	Checkup
iStdDev	Standard Deviation	Technical Indicators
iStochastic	Stochastic Oscillator	Technical Indicators
iTEMA	Triple Exponential Moving Average	Technical Indicators
iTriX	Triple Exponential Moving Averages Oscillator	Technical Indicators
iVIDyA	Variable Index Dynamic Average	Technical Indicators
iVolumes	Volumes	Technical Indicators
iWPR	Williams' Percent Range	Technical Indicators
log	Returns natural logarithm	Math Functions
log10	Returns the logarithm of a number by base 10	Math Functions
MarketBookAdd	Provides opening of Depth of Market for a selected symbol, and subscribes for receiving notifications of the DOM changes	Market Info

Function	Action	Section
MarketBookGet	Returns a structure array MqlBookInfo containing records of the Depth of Market of a specified symbol	Market Info
MarketBookRelease	Provides closing of Depth of Market for a selected symbol, and cancels the subscription for receiving notifications of the DOM changes	Market Info
MathAbs	Returns absolute value (modulus) of the specified numeric value	Math Functions
MathArccos	Returns the arc cosine of x in radians	Math Functions
MathArcsin	Returns the arc sine of x in radians	Math Functions
MathArctan	Returns the arc tangent of x in radians	Math Functions
MathCeil	Returns integer numeric value closest from above	Math Functions
MathCos	Returns the cosine of a number	Math Functions
MathExp	Returns exponent of a number	Math Functions
MathFloor	Returns integer numeric value closest from below	Math Functions
MathIsValidNumber	Checks the correctness of a real number	Math Functions
MathLog	Returns natural logarithm	Math Functions
MathLog10	Returns the logarithm of a number by base 10	Math Functions
MathMax	Returns the maximal value of the two numeric values	Math Functions
MathMin	Returns the minimal value of the two numeric values	Math Functions
MathMod	Returns the real remainder after the division of two numbers	Math Functions
MathPow	Raises the base to the specified power	Math Functions

Function	Action	Section
MathRand	Returns a pseudorandom value within the range of 0 to 32767	Math Functions
MathRound	Rounds of a value to the nearest integer	Math Functions
MathSin	Returns the sine of a number	Math Functions
MathSqrt	Returns a square root	Math Functions
MathSrand	Sets the starting point for generating a series of pseudorandom integers	Math Functions
MathTan	Returns the tangent of a number	Math Functions
MessageBox	Creates, displays a message box and manages it	Common Functions
MQLInfoInteger	Returns an integer value of a corresponding property of a running mql5 program	Checkup
MQLInfoString	Returns a string value of a corresponding property of a running mql5 program	Checkup
MT5Initialize	Establish a connection with the MetaTrader 5 terminal	MetaTrader for Python
MT5Shutdown	Close the previously established connection to the MetaTrader 5 terminal	MetaTrader for Python
MT5TerminalInfo	Get status and parameters of the connected MetaTrader 5 terminal	MetaTrader for Python
MT5Version	Return the MetaTrader 5 terminal version	MetaTrader for Python
MT5CopyRatesFrom	Get bars from the MetaTrader 5 terminal starting from the specified date	MetaTrader for Python
MT5CopyRatesFromPos	Get bars from the MetaTrader 5 terminal starting from the specified index	MetaTrader for Python
MT5CopyRatesRange	Get bars in the specified date range from the MetaTrader 5 terminal	MetaTrader for Python
MT5CopyTicksFrom	Get ticks from the MetaTrader 5 terminal starting from the	MetaTrader for Python

Function	Action	Section
	specified date	
MT5CopyTicksRange	Get ticks for the specified date range from the MetaTrader 5 terminal	MetaTrader for Python
NormalizeDouble	Rounding of a floating point number to a specified accuracy	Conversion Functions
ObjectCreate	Creates an object of the specified type in a specified chart	Object Functions
ObjectDelete	Removes the object with the specified name from the specified chart (from the specified chart subwindow)	Object Functions
ObjectFind	Searches for an object with the specified ID by the name	Object Functions
ObjectGetDouble	Returns the double value of the corresponding object property	Object Functions
ObjectGetInteger	Returns the integer value of the corresponding object property	Object Functions
ObjectGetString	Returns the string value of the corresponding object property	Object Functions
ObjectGetTimeByValue	Returns the time value for the specified object price value	Object Functions
ObjectGetValueByTime	Returns the price value of an object for the specified time	Object Functions
ObjectMove	Changes the coordinates of the specified object anchor point	Object Functions
ObjectName	Returns the name of an object of the corresponding type in the specified chart (specified chart subwindow)	Object Functions
ObjectsDeleteAll	Removes all objects of the specified type from the specified chart (from the specified chart subwindow)	Object Functions
ObjectSetDouble	Sets the value of the corresponding object property	Object Functions
ObjectSetInteger	Sets the value of the corresponding object property	Object Functions

Function	Action	Section
ObjectSetString	Sets the value of the corresponding object property	Object Functions
ObjectsTotal	Returns the number of objects of the specified type in the specified chart (specified chart subwindow)	Object Functions
OnStart	The function is called when the Start event occurs to perform actions set in the script	Event Handling
OnInit	The function is called in indicators and EAs when the Init event occurs to initialize a launched MQL5 program	Event Handling
OnDeinit	The function is called in indicators and EAs when the Deinit event occurs to de-initialize a launched MQL5 program	Event Handling
OnTick	The function is called in EAs when the NewTick event occurs to handle a new quote	Event Handling
OnCalculate	The function is called in indicators when the Calculate event occurs to handle price data changes	Event Handling
OnTimer	The function is called in indicators and EAs during the Timer periodic event generated by the terminal at fixed time intervals	Event Handling
OnTrade	The function is called in EAs during the Trade event generated at the end of a trading operation on a trade server	Event Handling
OnTradeTransaction	The function is called in EAs when the TradeTransaction event occurs to process a trade request execution results	Event Handling
OnBookEvent	The function is called in EAs when the BookEvent event	Event Handling

Function	Action	Section
	occurs to process changes in the market depth	
OnChartEvent	The function is called in indicators and EAs when the ChartEvent event occurs to process chart changes made by a user or an MQL5 program	Event Handling
OnTester	The function is called in EAs when the Tester event occurs to perform necessary actions after testing an EA on history data	Event Handling
OnTesterInit	The function is called in EAs when the TesterInit event occurs to perform necessary actions before optimization in the strategy tester	Event Handling
OnTesterDeinit	The function is called in EAs when the TesterDeinit event occurs after EA optimization in the strategy tester	Event Handling
OnTesterPass	The function is called in EAs when the TesterPass even occurs to handle an arrival of a new data frame during EA optimization in the strategy tester	Event Handling
OrderCalcMargin	Calculates the margin required for the specified order type, in the deposit currency	Trade Functions
OrderCalcProfit	Calculates the profit based on the parameters passed, in the deposit currency	Trade Functions
OrderCheck	Checks if there are enough funds to execute the required trade operation.	Trade Functions
OrderGetDouble	Returns the requested property of the order (double)	Trade Functions
OrderGetInteger	Returns the requested property of the order (datetime or int)	Trade Functions
OrderGetString	Returns the requested property of the order (string)	Trade Functions

Function	Action	Section
OrderGetTicket	Return the ticket of a corresponding order	Trade Functions
OrderSelect	Selects a order for further working with it	Trade Functions
OrderSend	Sends trade requests to a server	Trade Functions
OrderSendAsync	Asynchronously sends trade requests without waiting for the trade response of the trade server	Trade Functions
OrdersTotal	Returns the number of orders	Trade Functions
ParameterGetRange	Receives data on the values range and the change step for an input variable when optimizing an Expert Advisor in the Strategy Tester	Working with Optimization Results
ParameterSetRange	Specifies the use of input variable when optimizing an Expert Advisor in the Strategy Tester: value, change step, initial and final values	Working with Optimization Results
Period	Returns the current chart timeframe	Checkup
PeriodSeconds	Returns the number of seconds in the period	Common Functions
PlaySound	Plays a sound file	Common Functions
PlotIndexGetInteger	Returns the value of an indicator line property of the integer type	Custom Indicators
PlotIndexSetDouble	Sets the value of an indicator line property of the type double	Custom Indicators
PlotIndexSetInteger	Sets the value of an indicator line property of the int type	Custom Indicators
PlotIndexSetString	Sets the value of an indicator line property of the string type	Custom Indicators
Point	Returns the point size of the current symbol in the quote currency	Checkup
PositionGetDouble	Returns the requested property of an open position (double)	Trade Functions

Function	Action	Section
PositionGetInteger	Returns the requested property of an open position (datetime or int)	Trade Functions
PositionGetString	Returns the requested property of an open position (string)	Trade Functions
PositionGetSymbol	Returns the symbol corresponding to the open position	Trade Functions
PositionGetTicket	Returns the ticket of the position with the specified index in the list of open positions	Trade Functions
PositionSelect	Chooses an open position for further working with it	Trade Functions
PositionSelectByTicket	Selects a position to work with by the ticket number specified in it	Trade Functions
PositionsTotal	Returns the number of open positions	Trade Functions
pow	Raises the base to the specified power	Math Functions
Print	Displays a message in the log	Common Functions
PrintFormat	Formats and prints the sets of symbols and values in a log file in accordance with a preset format	Common Functions
rand	Returns a pseudorandom value within the range of 0 to 32767	Math Functions
ResetLastError	Sets the value of a predetermined variable _LastError to zero	Common Functions
ResourceCreate	Creates an image resource based on a data set	Common Functions
ResourceFree	Deletes dynamically created resource (freeing the memory allocated for it)	Common Functions
ResourceReadImage	Reads data from the graphical resource created by ResourceCreate() function or saved in EX5 file during compilation	Common Functions

Function	Action	Section
ResourceSave	Saves a resource into the specified file	Common Functions
round	Rounds of a value to the nearest integer	Math Functions
SendFTP	Sends a file at the address specified in the settings window of the "FTP" tab	Common Functions
SendMail	Sends an email at the address specified in the settings window of the "Email" tab	Common Functions
SendNotification	Sends push notifications to mobile terminals, whose MetaQuotes ID are specified in the "Notifications" tab	Common Functions
SeriesInfoInteger	Returns information about the state of historical data	Timeseries and Indicators Access
SetIndexBuffer	Binds the specified indicator buffer with one-dimensional dynamic array of the double type	Custom Indicators
ShortArrayToString	Copying array part into a string	Conversion Functions
ShortToString	Converting symbol code (unicode) into one-symbol string	Conversion Functions
SignalBaseGetDouble	Returns the value of double type property for selected signal	Trade Signals
SignalBaseGetInteger	Returns the value of integer type property for selected signal	Trade Signals
SignalBaseGetString	Returns the value of string type property for selected signal	Trade Signals
SignalBaseSelect	Selects a signal from signals, available in terminal for further working with it	Trade Signals
SignalBaseTotal	Returns the total amount of signals, available in terminal	Trade Signals
SignalInfoGetDouble	Returns the value of double type property of signal copy settings	Trade Signals
SignalInfoGetInteger	Returns the value of integer type property of signal copy settings	Trade Signals

Function	Action	Section
SignalInfoGetString	Returns the value of string type property of signal copy settings	Trade Signals
SignalInfoSetDouble	Sets the value of double type property of signal copy settings	Trade Signals
SignalInfoSetInteger	Sets the value of integer type property of signal copy settings	Trade Signals
SignalSubscribe	Subscribes to the trading signal	Trade Signals
SignalUnsubscribe	Cancels subscription	Trade Signals
sin	Returns the sine of a number	Math Functions
Sleep	Suspends execution of the current Expert Advisor or script within a specified interval	Common Functions
SocketCreate	Create a socket with specified flags and return its handle	Network Functions
SocketClose	Close a socket	Network Functions
SocketConnect	Connect to the server with timeout control	Network Functions
SocketIsConnected	Checks if the socket is currently connected	Network Functions
SocketIsReadable	Get a number of bytes that can be read from a socket	Network Functions
SocketIsWritable	Check whether data can be written to a socket at the current time	Network Functions
SocketTimeouts	Set timeouts for receiving and sending data for a socket system object	Network Functions
SocketRead	Read data from a socket	Network Functions
SocketSend	Write data to a socket	Network Functions
SocketTlsHandshake	Initiate secure TLS (SSL) connection to a specified host via TLS Handshake protocol	Network Functions
SocketTlsCertificate	Get data on the certificate used to secure network connection	Network Functions
SocketTlsRead	Read data from secure TLS connection	Network Functions

Function	Action	Section
SocketTlsReadAvailable	Read all available data from secure TLS connection	Network Functions
SocketTlsSend	Send data via secure TLS connection	Network Functions
sqrt	Returns a square root	Math Functions
srand	Sets the starting point for generating a series of pseudorandom integers	Math Functions
StringAdd	Adds a string to the end of another string	String Functions
StringBufferLen	Returns the size of buffer allocated for the string	String Functions
StringCompare	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1 (minus 1) - if the first string is less than the second one	String Functions
StringConcatenate	Forms a string of parameters passed	String Functions
StringFill	Fills out a specified string by selected symbols	String Functions
StringFind	Search for a substring in a string	String Functions
StringFormat	Converting number into string according to preset format	Conversion Functions
StringGetCharacter	Returns the value of a number located in the specified string position	String Functions
StringInit	Initializes string by specified symbols and provides the specified string length	String Functions
StringLen	Returns the number of symbols in a string	String Functions
StringReplace	Replaces all the found substrings of a string by a set sequence of symbols	String Functions
StringSetCharacter	Returns a copy of a string with a changed value of a symbol in a	String Functions

Function	Action	Section
	specified position	
StringSplit	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained	String Functions
StringSubstr	Extracts a substring from a text string starting from a specified position	String Functions
StringToCharArray	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type	Conversion Functions
StringToColor	Converting "R,G,B" string or string with color name into color type value	Conversion Functions
StringToDouble	Converting a string containing a symbol representation of number into number of double type	Conversion Functions
StringToInteger	Converting a string containing a symbol representation of number into number of int type	Conversion Functions
StringToLower	Transforms all symbols of a selected string to lowercase	String Functions
StringToShortArray	Symbol-wise copying a string to a selected part of array of ushort type	Conversion Functions
StringToTime	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type	Conversion Functions
StringToUpper	Transforms all symbols of a selected string into capitals	String Functions
StringTrimLeft	Cuts line feed characters, spaces and tabs in the left part of the string	String Functions
StringTrimRight	Cuts line feed characters, spaces and tabs in the right part of the string	String Functions
StructToTime	Converts a variable of MqlDateTime structure type into	Date and Time

Function	Action	Section
	a datetime value	
Symbol	Returns the name of a symbol of the current chart	Checkup
SymbolInfoDouble	Returns the double value of the symbol for the corresponding property	Market Info
SymbolInfoInteger	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property	Market Info
SymbolInfoMarginRate	Returns the margin rates depending on the order type and direction	Market Info
SymbolInfoSessionQuote	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.	Market Info
SymbolInfoSessionTrade	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.	Market Info
SymbolInfoString	Returns a value of the string type of a specified symbol for the corresponding property	Market Info
SymbolInfoTick	Returns the current prices for the specified symbol in a variable of the MqlTick type	Market Info
SymbolIsSynchronized	Checks whether data of a selected symbol in the terminal are synchronized with data on the trade server	Market Info
SymbolName	Returns the name of a specified symbol	Market Info
SymbolSelect	Selects a symbol in the Market Watch window or removes a symbol from the window	Market Info
SymbolsTotal	Returns the number of available (selected in Market Watch or all) symbols	Market Info

Function	Action	Section
tan	Returns the tangent of a number	Math Functions
TerminalClose	Commands the terminal to complete operation	Common Functions
TerminalInfoDouble	Returns an double value of a corresponding property of the mql5 program environment	Checkup
TerminalInfoInteger	Returns an integer value of a corresponding property of the mql5 program environment	Checkup
TerminalInfoString	Returns a string value of a corresponding property of the mql5 program environment	Checkup
TesterStatistics	It returns the value of a specified statistic calculated based on testing results	Common Functions
TextGetSize	Returns the string's width and height at the current font settings	Object Functions
TextOut	Transfers the text to the custom array (buffer) designed for creation of a graphical resource	Object Functions
TextSetFont	Sets the font for displaying the text using drawing methods (Arial 20 used by default)	Object Functions
TimeCurrent	Returns the last known server time (time of the last quote receipt) in the datetime format	Date and Time
TimeDaylightSavings	Returns the sign of Daylight Saving Time switch	Date and Time
TimeGMT	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running	Date and Time
TimeGMTOffset	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch	Date and Time
TimeLocal	Returns the local computer time in datetime format	Date and Time

Function	Action	Section
TimeToString	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format	Conversion Functions
TimeToStruct	Converts a datetime value into a variable of MqlDateTime structure type	Date and Time
TimeTradeServer	Returns the current calculation time of the trade server	Date and Time
UninitializeReason	Returns the code of the reason for deinitialization	Checkup
WebRequest	Sends HTTP request to the specified server	Common Functions
ZeroMemory	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.	Common Functions

List of MQL5 Constants

All MQL5 constants in alphabetical order.

Constant	Description	Usage
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)	Print
__DATETIME__	File compilation date and time	Print
__FILE__	Name of the currently compiled file	Print
__FUNCSIG__	Signature of the function in whose body the macro is located. Loggi	Print

Constant	Description	Usage
	ng of the full description of functions can be useful in the identification of overloaded functions	
__FUNCTION__	Name of the function, in whose body the macro is located	Print
__LINE__	Line number in the source code, in which the macro is located	Print
__MQLBUILD__, __MQL5BUILD__	Compiler	Print

Constant	Description	Usage
	build number	
__PATH__	An absolute path to the file that is currently being compiled	Print
ACCOUNT_ASSETS	The current assets of an account	AccountInfoDouble
ACCOUNT_BALANCE	Account balance in the deposit currency	AccountInfoDouble
ACCOUNT_COMMISSION_BLOCKED	The current blocked commission amount on an account	AccountInfoDouble

Constant	Description	Usage
ACCOUNT_COMPANY	Name of a company that serves the account	AccountInfoString
ACCOUNT_CREDIT	Account credit in the deposit currency	AccountInfoDouble
ACCOUNT_CURRENCY	Account currency	AccountInfoString
ACCOUNT_EQUITY	Account equity in the deposit currency	AccountInfoDouble
ACCOUNT_LEVERAGE	Account leverage	AccountInfoInteger
ACCOUNT_LIABILITIES	The current liabilities on an account	AccountInfoDouble
ACCOUNT_LIMIT_ORDERS	Maximum allowe	AccountInfoInteger

Constant	Description	Usage
	d number of active pending orders	
ACCOUNT_LOGIN	Account number	AccountInfoInteger
ACCOUNT_MARGIN	Account margin used in the deposit currency	AccountInfoDouble
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	AccountInfoDouble
ACCOUNT_MARGIN_INITIAL	Initial margin. The amount reserved on an account to cover the margin	AccountInfoDouble

Constant	Description	Usage
	of all pending orders	
ACCOUNT_MARGIN_LEVEL	Account margin level in percents	AccountInfoDouble
ACCOUNT_MARGIN_MAINTENANCE	Maintenance margin. The minimum equity reserved on an account to cover the minimum amount of all open positions	AccountInfoDouble
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is	AccountInfoDouble

Constant	Description	Usage
	expressed in percents or in the deposit currency	
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	AccountInfoInteger
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	AccountInfoDouble
ACCOUNT_NAME	Client name	AccountInfoString

Constant	Description	Usage
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	AccountInfoDouble
ACCOUNT_SERVER	Trade server name	AccountInfoString
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money	AccountInfoInteger
ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents	AccountInfoInteger
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	AccountInfoInteger
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert	AccountInfoInteger

Constant	Description	Usage
	Advisor	
ACCOUNT_TRADE_MODE	Account trade mode	AccountInfoInteger
ACCOUNT_TRADE_MODE_CONTEST	Contest account	AccountInfoInteger
ACCOUNT_TRADE_MODE_DEMO	Demo account	AccountInfoInteger
ACCOUNT_TRADE_MODE_REAL	Real account	AccountInfoInteger
ALIGN_CENTER	Centered (only for the Edit object)	ObjectSetInteger, ObjectGetInteger, ChartScreenShot
ALIGN_LEFT	Left alignment	ObjectSetInteger, ObjectGetInteger, ChartScreenShot
ALIGN_RIGHT	Right alignment	ObjectSetInteger, ObjectGetInteger, ChartScreenShot
ANCHOR_CENTER	Anchor point strictly in the center of the object	ObjectSetInteger, ObjectGetInteger
ANCHOR_LEFT	Anchor point	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	to the left in the center	
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner	ObjectSetInteger, ObjectGetInteger
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner	ObjectSetInteger, ObjectGetInteger
ANCHOR_LOWER	Anchor point below in the center	ObjectSetInteger, ObjectGetInteger
ANCHOR_RIGHT	Anchor point to the right in the center	ObjectSetInteger, ObjectGetInteger
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner	ObjectSetInteger, ObjectGetInteger
ANCHOR_RIGHT_UPPER	Anchor point at the upper	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	right corner	
ANCHOR_UPPER	Anchor point above in the center	ObjectSetInteger, ObjectGetInteger
BASE_LINE	Main line	Indicators Lines
BOOK_TYPE_BUY	Buy order (Bid)	MqlBookInfo
BOOK_TYPE_BUY_MARKET	Buy order by Market	MqlBookInfo
BOOK_TYPE_SELL	Sell order (Offer)	MqlBookInfo
BOOK_TYPE_SELL_MARKET	Sell order by Market	MqlBookInfo
BORDER_FLAT	Flat form	ObjectSetInteger, ObjectGetInteger
BORDER_RAISED	Prominent form	ObjectSetInteger, ObjectGetInteger
BORDER_SUNKEN	Concave form	ObjectSetInteger, ObjectGetInteger
CHAR_MAX	Maximal value, which can be	Numerical Type Constants

Constant	Description	Usage
	represented by char type	
CHAR_MIN	Minimal value, which can be represented by char type	Numerical Type Constants
CHART_AUTOSCROLL	Mode of automatic moving to the right border of the chart	ChartSetInteger, ChartGetInteger
CHART_BARS	Display as a sequence of bars	ChartSetInteger
CHART_BEGIN	Chart beginning (the oldest prices)	ChartNavigate
CHART_BRING_TO_TOP	Show chart on top of	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	other charts	
CHART_CANDLES	Display as Japanese candlesticks	ChartSetInteger
CHART_COLOR_ASK	Ask price level color	ChartSetInteger, ChartGetInteger
CHART_COLOR_BACKGROUND	Chart background color	ChartSetInteger, ChartGetInteger
CHART_COLOR_BID	Bid price level color	ChartSetInteger, ChartGetInteger
CHART_COLOR_CANDLE_BEAR	Body color of a bear candlestick	ChartSetInteger, ChartGetInteger
CHART_COLOR_CANDLE_BULL	Body color of a bull candlestick	ChartSetInteger, ChartGetInteger
CHART_COLOR_CHART_DOWN	Color for the down bar, shadows and body borders of	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	bear candlesticks	
CHART_COLOR_CHART_LINE	Line chart color and color of "Doji" Japanese candlesticks	ChartSetInteger, ChartGetInteger
CHART_COLOR_CHART_UP	Color for the up bar, shadows and body borders of bull candlesticks	ChartSetInteger, ChartGetInteger
CHART_COLOR_FOREGROUND	Color of axes, scales and OHLC line	ChartSetInteger, ChartGetInteger
CHART_COLOR_GRID	Grid color	ChartSetInteger, ChartGetInteger
CHART_COLOR_LAST	Line color of the last executed deal price (Last)	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
CHART_COLOR_STOP_LEVEL	Color of stop order levels (Stop Loss and Take Profit)	ChartSetInteger, ChartGetInteger
CHART_COLOR_VOLUME	Color of volumes and position opening levels	ChartSetInteger, ChartGetInteger
CHART_COMMENT	Text of a comment in a chart	ChartSetString, ChartGetString
CHART_CURRENT_POS	Current position	ChartNavigate
CHART_DRAG_TRADE_LEVELS	Permission to drag trading levels on a chart with a mouse. The drag	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	mode is enabled by default (true value)	
CHART_EVENT_MOUSE_MOVE	Send notifications of mouse move and mouse click events (CHARTEVENT_MOUSE_MOVE) to all mql5 programs on a chart	ChartSetInteger, ChartGetInteger
CHART_EVENT_OBJECT_CREATE	Send a notification of an event of new object creation (CHARTEVENT_O	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	BJECT_CREATE) to all mql5-programs on a chart	
CHART_EVENT_OBJECT_DELETE	Send a notification of an event of object deletion (CHARTEVENT_OBJECT_DELETE) to all mql5-programs on a chart	ChartSetInteger, ChartGetInteger
CHART_FIRST_VISIBLE_BAR	Number of the first visible bar in the chart. Indexing of bars is the same as for	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	timeseries.	
CHART_FIXED_MAX	Fixed chart maximum	ChartSetDouble, ChartGetDouble
CHART_FIXED_MIN	Fixed chart minimum	ChartSetDouble, ChartGetDouble
CHART_FIXED_POSITION	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the horizontal time axis. It is displayed only if the automatic chart scrolli	ChartSetDouble, ChartGetDouble

Constant	Description	Usage
	ng to the right on tick incoming is disabled (see CHART_AUTOSCROLL property). The bar on a fixed position remains in the same place when zooming in and out.	
CHART_FOREGROUND	Price chart in the foreground	ChartSetInteger, ChartGetInteger
CHART_HEIGHT_IN_PIXELS	Chart height in pixels	ChartSetInteger, ChartGetInteger
CHART_IS_OBJECT	Identifying "Chart" (OBJ_CHA	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	RT) object - returns true for a graphical object. Returns false for a real chart	
CHART_LINE	Display as a line drawn by Close prices	ChartSetInteger
CHART_MODE	Chart type (candlesticks, bars or line)	ChartSetInteger, ChartGetInteger
CHART_MOUSE_SCROLL	Scrolling the chart horizontally using the left mouse button. Vertical scrolli	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	ng is also available if the value of any following properties is set to true: CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR	
CHART_POINTS_PER_BAR	Scale in points per bar	ChartSetDouble, ChartGetDouble
CHART_PRICE_MAX	Chart maximum	ChartSetDouble, ChartGetDouble
CHART_PRICE_MIN	Chart minimum	ChartSetDouble, ChartGetDouble
CHART_SCALE	Scale	ChartSetInteger, ChartGetInteger
CHART_SCALE_PT_PER_BAR	Scale to be specified in points	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	per bar	
CHART_SCALEFIX	Fixed scale mode	ChartSetInteger, ChartGetInteger
CHART_SCALEFIX_11	Scale 1:1 mode	ChartSetInteger, ChartGetInteger
CHART_SHIFT	Mode of price chart indent from the right border	ChartSetInteger, ChartGetInteger
CHART_SHIFT_SIZE	The size of the zero bar indent from the right border in percents	ChartSetDouble, ChartGetDouble
CHART_SHOW_ASK_LINE	Display Ask values as a horizontal line in a chart	ChartSetInteger, ChartGetInteger
CHART_SHOW_BID_LINE	Display Bid values as a	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	horizontal line in a chart	
CHART_SHOW_DATE_SCALE	Showing the time scale on a chart	ChartSetInteger, ChartGetInteger
CHART_SHOW_GRID	Display grid in the chart	ChartSetInteger, ChartGetInteger
CHART_SHOW_LAST_LINE	Display Last values as a horizontal line in a chart	ChartSetInteger, ChartGetInteger
CHART_SHOW_OBJECT_DESCR	Pop-up descriptions of graphical objects	ChartSetInteger, ChartGetInteger
CHART_SHOW_OHLC	Show OHLC values in the upper left corner	ChartSetInteger, ChartGetInteger
CHART_SHOW_ONE_CLICK	Showing the "One click	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	trading" panel on a chart	
CHART_SHOW_PERIOD_SEP	Display vertical separators between adjacent periods	ChartSetInteger, ChartGetInteger
CHART_SHOW_PRICE_SCALE	Showing the price scale on a chart	ChartSetInteger, ChartGetInteger
CHART_SHOW_TRADE_LEVELS	Displaying trade levels in the chart (levels of open positions, Stop Loss, Take Profit and pending orders)	ChartSetInteger, ChartGetInteger
CHART_SHOW_VOLUMES	Display	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	volume in the chart	
CHART_VISIBLE_BARS	The number of bars on the chart that can be displayed	ChartSetInteger, ChartGetInteger
CHART_VOLUME_HIDE	Volumes are not shown	ChartSetInteger
CHART_VOLUME_REAL	Trade volumes	ChartSetInteger
CHART_VOLUME_TICK	Tick volumes	ChartSetInteger
CHART_WIDTH_IN_BARS	Chart width in bars	ChartSetInteger, ChartGetInteger
CHART_WIDTH_IN_PIXELS	Chart width in pixels	ChartSetInteger, ChartGetInteger
CHART_WINDOW_HANDLE	Chart window handle (HWND)	ChartSetInteger, ChartGetInteger
CHART_WINDOW_IS_VISIBLE	Visibility of	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	subwindows	
CHART_WINDOW_YDISTANCE	The distance between the upper frame of the indicator subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a mouse event, the cursor coordinates are passed in terms of the coordinates	ChartSetInteger, ChartGetInteger

Constant	Description	Usage
	of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow. The value is required for converting the absolute coordinates of the main chart to the local	

Constant	Description	Usage
	coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.	
CHART_WINDOWS_TOTAL	The total number of chart windows, including indicator subwindows	ChartSetInteger, ChartGetInteger
CHARTEVENT_CHART_CHANGE	Change of the	OnChartEvent

Constant	Description	Usage
	chart size or modification of chart properties through the Properties dialog	
CHARTEVENT_CLICK	Clicking on a chart	OnChartEvent
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events	OnChartEvent
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events	OnChartEvent

Constant	Description	Usage
CHARTEVENT_KEYDOWN	Keystrokes	OnChartEvent
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)	OnChartEvent
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog	OnChartEvent
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object	OnChartEvent
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT	OnChartEvent

Constant	Description	Usage
	_CREATE=true is set for the chart)	
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	OnChartEvent
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object	OnChartEvent
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit	OnChartEvent
CHARTS_MAX	The maximum possible numb	Other Constants

Constant	Description	Usage
	er of simultaneously open charts in the terminal	
CHIKOUSPAN_LINE	Chikou Span line	Indicators Lines
clrAliceBlue	Alice Blue	Web Colors
clrAntiqueWhite	Antique White	Web Colors
clrAqua	Aqua	Web Colors
clrAquamarine	Aquamarine	Web Colors
clrBeige	Beige	Web Colors
clrBisque	Bisque	Web Colors
clrBlack	Black	Web Colors
clrBlanchedAlmond	Blanched Almond	Web Colors
clrBlue	Blue	Web Colors
clrBlueViolet	Blue Violet	Web Colors
clrBrown	Brown	Web Colors
clrBurlyWood	Burly Wood	Web Colors
clrCadetBlue	Cadet Blue	Web Colors

Constant	Description	Usage
clrChartreuse	Chartreuse	Web Colors
clrChocolate	Chocolate	Web Colors
clrCoral	Coral	Web Colors
clrCornflowerBlue	Cornflower Blue	Web Colors
clrCornsilk	Cornsilk	Web Colors
clrCrimson	Crimson	Web Colors
clrDarkBlue	Dark Blue	Web Colors
clrDarkGoldenrod	Dark Goldenrod	Web Colors
clrDarkGray	Dark Gray	Web Colors
clrDarkGreen	Dark Green	Web Colors
clrDarkKhaki	Dark Khaki	Web Colors
clrDarkOliveGreen	Dark Olive Green	Web Colors
clrDarkOrange	Dark Orange	Web Colors
clrDarkOrchid	Dark Orchid	Web Colors
clrDarkSalmon	Dark Salmon	Web Colors
clrDarkSeaGreen	Dark Sea Green	Web Colors

Constant	Description	Usage
clrDarkSlateBlue	Dark Slate Blue	Web Colors
clrDarkSlateGray	Dark Slate Gray	Web Colors
clrDarkTurquoise	Dark Turquoise	Web Colors
clrDarkViolet	Dark Violet	Web Colors
clrDeepPink	Deep Pink	Web Colors
clrDeepSkyBlue	Deep Sky Blue	Web Colors
clrDimGray	Dim Gray	Web Colors
clrDodgerBlue	Dodger Blue	Web Colors
clrFireBrick	Fire Brick	Web Colors
clrForestGreen	Forest Green	Web Colors
clrGainsboro	Gainsboro	Web Colors
clrGold	Gold	Web Colors
clrGoldenrod	Goldenrod	Web Colors
clrGray	Gray	Web Colors
clrGreen	Green	Web Colors
clrGreenYellow	Green Yellow	Web Colors
clrHoneydew	Honeydew	Web Colors
clrHotPink	Hot Pink	Web Colors

Constant	Description	Usage
clrIndianRed	Indian Red	Web Colors
clrIndigo	Indigo	Web Colors
clrIvory	Ivory	Web Colors
clrKhaki	Khaki	Web Colors
clrLavender	Lavender	Web Colors
clrLavenderBlush	Lavender Blush	Web Colors
clrLawnGreen	Lawn Green	Web Colors
clrLemonChiffon	Lemon Chiffon	Web Colors
clrLightBlue	Light Blue	Web Colors
clrLightCoral	Light Coral	Web Colors
clrLightCyan	Light Cyan	Web Colors
clrLightGoldenrod	Light Goldenrod	Web Colors
clrLightGray	Light Gray	Web Colors
clrLightGreen	Light Green	Web Colors
clrLightPink	Light Pink	Web Colors
clrLightSalmon	Light Salmon	Web Colors
clrLightSeaGreen	Light Sea Green	Web Colors

Constant	Description	Usage
clrLightSkyBlue	Light Sky Blue	Web Colors
clrLightSlateGray	Light Slate Gray	Web Colors
clrLightSteelBlue	Light Steel Blue	Web Colors
clrLightYellow	Light Yellow	Web Colors
clrLime	Lime	Web Colors
clrLimeGreen	Lime Green	Web Colors
clrLinen	Linen	Web Colors
clrMagenta	Magenta	Web Colors
clrMaroon	Maroon	Web Colors
clrMediumAquamarine	Medium Aquamarine	Web Colors
clrMediumBlue	Medium Blue	Web Colors
clrMediumOrchid	Medium Orchid	Web Colors
clrMediumPurple	Medium Purple	Web Colors
clrMediumSeaGreen	Medium Sea Green	Web Colors
clrMediumSlateBlue	Medium	Web Colors

Constant	Description	Usage
	Slate Blue	
clrMediumSpringGreen	Medium Spring Green	Web Colors
clrMediumTurquoise	Medium Turquoise	Web Colors
clrMediumVioletRed	Medium Violet Red	Web Colors
clrMidnightBlue	Midnight Blue	Web Colors
clrMintCream	Mint Cream	Web Colors
clrMistyRose	Misty Rose	Web Colors
clrMoccasin	Moccasin	Web Colors
clrNavajoWhite	Navajo White	Web Colors
clrNavy	Navy	Web Colors
clrNONE	Absence of color	Other Constants
clrOldLace	Old Lace	Web Colors
clrOlive	Olive	Web Colors
clrOliveDrab	Olive Drab	Web Colors
clrOrange	Orange	Web Colors

Constant	Description	Usage
clrOrangeRed	Orange Red	Web Colors
clrOrchid	Orchid	Web Colors
clrPaleGoldenrod	Pale Goldenrod	Web Colors
clrPaleGreen	Pale Green	Web Colors
clrPaleTurquoise	Pale Turquoise	Web Colors
clrPaleVioletRed	Pale Violet Red	Web Colors
clrPapayaWhip	Papaya Whip	Web Colors
clrPeachPuff	Peach Puff	Web Colors
clrPeru	Peru	Web Colors
clrPink	Pink	Web Colors
clrPlum	Plum	Web Colors
clrPowderBlue	Powder Blue	Web Colors
clrPurple	Purple	Web Colors
clrRed	Red	Web Colors
clrRosyBrown	Rosy Brown	Web Colors
clrRoyalBlue	Royal Blue	Web Colors
clrSaddleBrown	Saddle Brown	Web Colors
clrSalmon	Salmon	Web Colors

Constant	Description	Usage
clrSandyBrown	Sandy Brown	Web Colors
clrSeaGreen	Sea Green	Web Colors
clrSeashell	Seashell	Web Colors
clrSienna	Sienna	Web Colors
clrSilver	Silver	Web Colors
clrSkyBlue	Sky Blue	Web Colors
clrSlateBlue	Slate Blue	Web Colors
clrSlateGray	Slate Gray	Web Colors
clrSnow	Snow	Web Colors
clrSpringGreen	Spring Green	Web Colors
clrSteelBlue	Steel Blue	Web Colors
clrTan	Tan	Web Colors
clrTeal	Teal	Web Colors
clrThistle	Thistle	Web Colors
clrTomato	Tomato	Web Colors
clrTurquoise	Turquoise	Web Colors
clrViolet	Violet	Web Colors
clrWheat	Wheat	Web Colors
clrWhite	White	Web Colors
clrWhiteSmoke	White Smoke	Web Colors

Constant	Description	Usage
clrYellow	Yellow	Web Colors
clrYellowGreen	Yellow Green	Web Colors
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart	ObjectSetInteger, ObjectGetInteger
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart	ObjectSetInteger, ObjectGetInteger
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart	ObjectSetInteger, ObjectGetInteger
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	of the chart	
CP_ACP	The current Windows ANSI code page.	CharArrayToString, StringToCharArray, FileOpen
CP_MACCP	The current system Macintosh code page. Note: This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unico	CharArrayToString, StringToCharArray, FileOpen

Constant	Description	Usage
	de for encoding.	
CP_OEMCP	The current system OEM code page.	CharArrayToString, StringToCharArray, FileOpen
CP_SYMBOL	Symbol code page	CharArrayToString, StringToCharArray, FileOpen
CP_THREAD_ACP	The Windows ANSI code page for the current thread.	CharArrayToString, StringToCharArray, FileOpen
CP_UTF7	UTF-7 code page.	CharArrayToString, StringToCharArray, FileOpen
CP_UTF8	UTF-8 code page.	CharArrayToString, StringToCharArray, FileOpen
CRYPT_AES128	AES encryption with 128 bit key (16 bytes)	CryptEncode, CryptDecode

Constant	Description	Usage
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)	CryptEncode, CryptDecode
CRYPT_ARCH_ZIP	ZIP archives	CryptEncode, CryptDecode
CRYPT_BASE64	BASE64	CryptEncode, CryptDecode
CRYPT_DES	DES encryption with 56 bit key (7 bytes)	CryptEncode, CryptDecode
CRYPT_HASH_MD5	MD5 HASH calculation	CryptEncode, CryptDecode
CRYPT_HASH_SHA1	SHA1 HASH calculation	CryptEncode, CryptDecode
CRYPT_HASH_SHA256	SHA256 HASH calculation	CryptEncode, CryptDecode
DBL_DIG	Number of significant decimal digits for	Numerical Type Constants

Constant	Description	Usage
	double type	
DBL_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for double type)	Numerical Type Constants
DBL_MANT_DIG	Bits count in a mantissa for double type	Numerical Type Constants
DBL_MAX	Maximal value, which can be represented by double type	Numerical Type Constants
DBL_MAX_10_EXP	Maximal decimal value of exponent	Numerical Type Constants

Constant	Description	Usage
	degree for double type	
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	Numerical Type Constants
DBL_MIN	Minimal positive value, which can be represented by double type	Numerical Type Constants
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	Numerical Type Constants
DBL_MIN_EXP	Minimal binary value of	Numerical Type Constants

Constant	Description	Usage
	exponent degree for double type	
DEAL_COMMENT	Deal comment	HistoryDealGetString
DEAL_COMMISSION	Deal commission	HistoryDealGetDouble
DEAL_ENTRY	Deal entry - entry in, entry out, reverse	HistoryDealGetInteger
DEAL_ENTRY_IN	Entry in	HistoryDealGetInteger
DEAL_ENTRY_INOUT	Reverse	HistoryDealGetInteger
DEAL_ENTRY_OUT	Entry out	HistoryDealGetInteger
DEAL_MAGIC	Deal magic number (see ORDER_MAGIC)	HistoryDealGetInteger
DEAL_ORDER	Deal order number	HistoryDealGetInteger
DEAL_POSITION_ID	Identifier of a	HistoryDealGetInteger

Constant	Description	Usage
	position, in the opening, modification or change of which this deal took part. Each position has a unique identifier that is assigned to all deals executed for the symbol during the entire lifetime of the position.	
DEAL_PRICE	Deal price	HistoryDealGetDouble

Constant	Description	Usage
DEAL_PROFIT	Deal profit	HistoryDealGetDouble
DEAL_SWAP	Cumulative swap on close	HistoryDealGetDouble
DEAL_SYMBOL	Deal symbol	HistoryDealGetString
DEAL_TIME	Deal time	HistoryDealGetInteger
DEAL_TIME_MSC	The time of a deal execution in milliseconds since 01.01.1970	HistoryDealGetInteger
DEAL_TYPE	Deal type	HistoryDealGetInteger
DEAL_TYPE_BALANCE	Balance	HistoryDealGetInteger
DEAL_TYPE_BONUS	Bonus	HistoryDealGetInteger
DEAL_TYPE_BUY	Buy	HistoryDealGetInteger
DEAL_TYPE_BUY_CANCELED	Cancelled buy deal. There can be a situation when a	HistoryDealGetInteger

Constant	Description	Usage
	previously executed buy deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_BUY) is changed to DEAL_TYPE_BUY_CANCELED, and its profit/loss is zeroized. Previously obtained profit/loss is charged/withdrawn	

Constant	Description	Usage
	using a separated balance operation	
DEAL_TYPE_CHARGE	Additional charge	HistoryDealGetInteger
DEAL_TYPE_COMMISSION	Additional commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_AGENT_DAILY	Daily agent commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Monthly agent commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_DAILY	Daily commission	HistoryDealGetInteger
DEAL_TYPE_COMMISSION_MONTHLY	Monthly commission	HistoryDealGetInteger
DEAL_TYPE_CORRECTION	Correction	HistoryDealGetInteger
DEAL_TYPE_CREDIT	Credit	HistoryDealGetInteger
DEAL_TYPE_INTEREST	Interest rate	HistoryDealGetInteger
DEAL_TYPE_SELL	Sell	HistoryDealGetInteger
DEAL_TYPE_SELL_CANCELED	Canceled	HistoryDealGetInteger

Constant	Description	Usage
	sell deal. There can be a situation when a previously executed sell deal is canceled. In this case, the type of the previously executed deal (DEAL_TYPE_SELL) is changed to DEAL_TYPE_SELL_CANCELED, and its profit/loss is zeroized. Previously	

Constant	Description	Usage
	obtained profit/loss is charged/withdrawn using a separated balance operation	
DEAL_VOLUME	Deal volume	HistoryDealGetDouble
DRAW_ARROW	Drawing arrows	Drawing Styles
DRAW_BARS	Display as a sequence of bars	Drawing Styles
DRAW_CANDLES	Display as a sequence of candlesticks	Drawing Styles
DRAW_COLOR_ARROW	Drawing multicolored arrows	Drawing Styles
DRAW_COLOR_BARS	Multicolored bars	Drawing Styles

Constant	Description	Usage
DRAW_COLOR_CANDLES	Multicolored candlesticks	Drawing Styles
DRAW_COLOR_HISTOGRAM	Multicolored histogram from the zero line	Drawing Styles
DRAW_COLOR_HISTOGRAM2	Multicolored histogram of the two indicator buffers	Drawing Styles
DRAW_COLOR_LINE	Multicolored line	Drawing Styles
DRAW_COLOR_SECTION	Multicolored section	Drawing Styles
DRAW_COLOR_ZIGZAG	Multicolored ZigZag	Drawing Styles
DRAW_FILLING	Color fill between the two levels	Drawing Styles
DRAW_HISTOGRAM	Histogram from the	Drawing Styles

Constant	Description	Usage
	zero line	
DRAW_HISTOGRAM2	Histogram of the two indicator buffers	Drawing Styles
DRAW_LINE	Line	Drawing Styles
DRAW_NONE	Not drawn	Drawing Styles
DRAW_SECTION	Section	Drawing Styles
DRAW_ZIGZAG	Style Zigzag allows vertical section on the bar	Drawing Styles
ELLIOTT_CYCLE	Cycle	ObjectSetInteger, ObjectGetInteger
ELLIOTT_GRAND_SUPERCYCLE	Grand Supercycle	ObjectSetInteger, ObjectGetInteger
ELLIOTT_INTERMEDIATE	Intermediate	ObjectSetInteger, ObjectGetInteger
ELLIOTT_MINOR	Minor	ObjectSetInteger, ObjectGetInteger
ELLIOTT_MINUETTE	Minuette	ObjectSetInteger, ObjectGetInteger
ELLIOTT_MINUTE	Minute	ObjectSetInteger, ObjectGetInteger
ELLIOTT_PRIMARY	Primary	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
ELLIOTT_SUBMINUETTE	Subminuette	ObjectSetInteger, ObjectGetInteger
ELLIOTT_SUPERCYCLE	Supercycle	ObjectSetInteger, ObjectGetInteger
EMPTY_VALUE	Empty value in an indicator buffer	Other Constants
ERR_ACCOUNT_WRONG_PROPERTY	Wrong account property ID	GetLastError
ERR_ARRAY_BAD_SIZE	Requested array size exceeds 2 GB	GetLastError
ERR_ARRAY_RESIZE_ERROR	Not enough memory for the relocation of an array, or an attempt to change the size of a static array	GetLastError

Constant	Description	Usage
ERR_BOOKS_CANNOT_ADD	Depth Of Market can not be added	GetLastError
ERR_BOOKS_CANNOT_DELETE	Depth Of Market can not be removed	GetLastError
ERR_BOOKS_CANNOT_GET	The data from Depth Of Market can not be obtained	GetLastError
ERR_BOOKS_CANNOT_SUBSCRIBE	Error in subscribing to receive new data from Depth Of Market	GetLastError
ERR_BUFFERS_NO_MEMORY	Not enough memory for the distribution of	GetLastError

Constant	Description	Usage
	indicator buffers	
ERR_BUFFERS_WRONG_INDEX	Wrong indicator buffer index	GetLastError
ERR_CANNOT_CLEAN_DIRECTORY	Failed to clear the directory (probably one or more files are blocked and removal operation failed)	GetLastError
ERR_CANNOT_DELETE_DIRECTORY	The directory cannot be removed	GetLastError
ERR_CANNOT_DELETE_FILE	File deleting error	GetLastError
ERR_CANNOT_OPEN_FILE	File opening error	GetLastError

Constant	Description	Usage
ERR_CHAR_ARRAY_ONLY	Must be an array of type char	GetLastError
ERR_CHART_CANNOT_CHANGE	Failed to change chart symbol and period	GetLastError
ERR_CHART_CANNOT_CREATE_TIMER	Failed to create timer	GetLastError
ERR_CHART_CANNOT_OPEN	Chart opening error	GetLastError
ERR_CHART_INDICATOR_CANNOT_ADD	Error adding an indicator to chart	GetLastError
ERR_CHART_INDICATOR_CANNOT_DEL	Error deleting an indicator from the chart	GetLastError
ERR_CHART_INDICATOR_NOT_FOUND	Indicator not found on the specified chart	GetLastError

Constant	Description	Usage
ERR_CHART_NAVIGATE_FAILED	Error navigating through chart	GetLastError
ERR_CHART_NO_EXPERT	No Expert Advisor in the chart that could handle the event	GetLastError
ERR_CHART_NO_REPLY	Chart does not respond	GetLastError
ERR_CHART_NOT_FOUND	Chart not found	GetLastError
ERR_CHART_SCREENSHOT_FAILED	Error creating screenshots	GetLastError
ERR_CHART_TEMPLATE_FAILED	Error applying template	GetLastError
ERR_CHART_WINDOW_NOT_FOUND	Subwindow containing the indicator	GetLastError

Constant	Description	Usage
	was not found	
ERR_CHART_WRONG_ID	Wrong chart ID	GetLastError
ERR_CHART_WRONG_PARAMETER	Error value of the parameter for the function of working with charts	GetLastError
ERR_CHART_WRONG_PROPERTY	Wrong chart property ID	GetLastError
ERR_CUSTOM_WRONG_PROPERTY	Wrong ID of the custom indicator property	GetLastError
ERR_DIRECTORY_NOT_EXIST	Directory does not exist	GetLastError
ERR_DOUBLE_ARRAY_ONLY	Must be an array of type	GetLastError

Constant	Description	Usage
	double	
ERR_FILE_BINSTRINGSIZE	String size must be specified, because the file is opened as binary	GetLastError
ERR_FILE_CACHEBUFFER_ERROR	Not enough memory for cache to read	GetLastError
ERR_FILE_CANNOT_REWRITE	File can not be rewritten	GetLastError
ERR_FILE_IS_DIRECTORY	This is not a file, this is a directory	GetLastError
ERR_FILE_ISNOT_DIRECTORY	This is a file, not a directory	GetLastError
ERR_FILE_NOT_EXIST	File does not exist	GetLastError

Constant	Description	Usage
ERR_FILE_NOTBIN	The file must be opened as a binary one	GetLastError
ERR_FILE_NOTCSV	The file must be opened as CSV	GetLastError
ERR_FILE_NOTTOREAD	The file must be opened for reading	GetLastError
ERR_FILE_NOTTOWRITE	The file must be opened for writing	GetLastError
ERR_FILE_NOTTXT	The file must be opened as a text	GetLastError
ERR_FILE_NOTTXTORCSV	The file must be opened as a text	GetLastError

Constant	Description	Usage
	or CSV	
ERR_FILE_READERROR	File reading error	GetLastError
ERR_FILE_WRITEERROR	Failed to write a resource to a file	GetLastError
ERR_FLOAT_ARRAY_ONLY	Must be an array of type float	GetLastError
ERR_FTP_SEND_FAILED	File sending via ftp failed	GetLastError
ERR_FUNCTION_NOT_ALLOWED	Function is not allowed for call	GetLastError
ERR_GLOBALVARIABLE_EXISTS	Global variable of the client terminal with the same name already exists	GetLastError

Constant	Description	Usage
ERR_GLOBALVARIABLE_NOT_FOUND	Global variable of the client terminal is not found	GetLastError
ERR_HISTORY_NOT_FOUND	Requested history not found	GetLastError
ERR_HISTORY_WRONG_PROPERTY	Wrong ID of the history property	GetLastError
ERR_INCOMPATIBLE_ARRAYS	Copying incompatible arrays. String array can be copied only to a string array, and a numeric array - in numeric array only	GetLastError

Constant	Description	Usage
ERR_INCOMPATIBLE_FILE	A text file must be for string arrays, for other arrays - binary	GetLastError
ERR_INDICATOR_CANNOT_ADD	Error applying an indicator to chart	GetLastError
ERR_INDICATOR_CANNOT_APPLY	The indicator cannot be applied to another indicator	GetLastError
ERR_INDICATOR_CANNOT_CREATE	Indicator cannot be created	GetLastError
ERR_INDICATOR_CUSTOM_NAME	The first parameter in the array must be the name of the custo	GetLastError

Constant	Description	Usage
	m indicator	
ERR_INDICATOR_DATA_NOT_FOUND	Requested data not found	GetLastError
ERR_INDICATOR_NO_MEMORY	Not enough memory to add the indicator	GetLastError
ERR_INDICATOR_PARAMETER_TYPE	Invalid parameter type in the array when creating an indicator	GetLastError
ERR_INDICATOR_PARAMETERS_MISSING	No parameters when creating an indicator	GetLastError
ERR_INDICATOR_UNKNOWN_SYMBOL	Unknown symbol	GetLastError
ERR_INDICATOR_WRONG_HANDLE	Wrong indica	GetLastError

Constant	Description	Usage
	tor handle	
ERR_INDICATOR_WRONG_INDEX	Wrong index of the requested indicator buffer	GetLastError
ERR_INDICATOR_WRONG_PARAMETERS	Wrong number of parameters when creating an indicator	GetLastError
ERR_INT_ARRAY_ONLY	Must be an array of type int	GetLastError
ERR_INTERNAL_ERROR	Unexpected internal error	GetLastError
ERR_INVALID_ARRAY	Array of a wrong type, wrong size, or a damaged object of a	GetLastError

Constant	Description	Usage
	dynamic array	
ERR_INVALID_DATETIME	Invalid date and/or time	GetLastError
ERR_INVALID_FILEHANDLE	A file with this handle was closed, or was not opening at all	GetLastError
ERR_INVALID_PARAMETER	Wrong parameter when calling the system function	GetLastError
ERR_INVALID_POINTER	Wrong pointer	GetLastError
ERR_INVALID_POINTER_TYPE	Wrong type of pointer	GetLastError
ERR_LONG_ARRAY_ONLY	Must be an array of	GetLastError

Constant	Description	Usage
	type long	
ERR_MAIL_SEND_FAILED	Email sending failed	GetLastError
ERR_MARKET_LASTTIME_UNKNOWN	Time of the last tick is not known (no ticks)	GetLastError
ERR_MARKET_NOT_SELECTED	Symbol is not selected in MarketWatch	GetLastError
ERR_MARKET_SELECT_ERROR	Error adding or deleting a symbol in MarketWatch	GetLastError
ERR_MARKET_UNKNOWN_SYMBOL	Unknown symbol	GetLastError
ERR_MARKET_WRONG_PROPERTY	Wrong identifier of a symbol	GetLastError

Constant	Description	Usage
	property	
ERR_MQL5_WRONG_PROPERTY	Wrong identifier of the program property	GetLastError
ERR_NO_STRING_DATE	No date in the string	GetLastError
ERR_NOT_ENOUGH_MEMORY	Not enough memory to perform the system function	GetLastError
ERR_NOTIFICATION_SEND_FAILED	Failed to send a notification	GetLastError
ERR_NOTIFICATION_TOO_FREQUENT	Too frequent sending of notifications	GetLastError
ERR_NOTIFICATION_WRONG_PARAMETER	Invalid parameter	GetLastError

Constant	Description	Usage
	for sending a notification – an empty string or NULL has been passed to the SendNotification() function	
ERR_NOTIFICATION_WRONG_SETTINGS	Wrong settings of notifications in the terminal (ID is not specified or permission is not set)	GetLastError
ERR_NOTINITIALIZED_STRING	Not initialized string	GetLastError
ERR_NUMBER_ARRAYS_ONLY	Must be a numer	GetLastError

Constant	Description	Usage
	ic array	
ERR_OBJECT_ERROR	Error working with a graphical object	GetLastError
ERR_OBJECT_GETDATE_FAILED	Unable to get date corresponding to the value	GetLastError
ERR_OBJECT_GETVALUE_FAILED	Unable to get value corresponding to the date	GetLastError
ERR_OBJECT_NOT_FOUND	Graphical object was not found	GetLastError
ERR_OBJECT_WRONG_PROPERTY	Wrong ID of a graphical object property	GetLastError
ERR_ONEDIM_ARRAYS_ONLY	Must be a one-	GetLastError

Constant	Description	Usage
	dimensional array	
ERR_OPENCL_BUFFER_CREATE	Failed to create an OpenCL buffer	GetLastError
ERR_OPENCL_CONTEXT_CREATE	Error creating the OpenCL context	GetLastError
ERR_OPENCL_EXECUTE	OpenCL program runtime error	GetLastError
ERR_OPENCL_INTERNAL	Internal error occurred when running OpenCL	GetLastError
ERR_OPENCL_INVALID_HANDLE	Invalid OpenCL handle	GetLastError
ERR_OPENCL_KERNEL_CREATE	Error creating an Open	GetLastError

Constant	Description	Usage
	CL kernel	
ERR_OPENCL_NOT_SUPPORTED	OpenCL functions are not supported on this computer	GetLastError
ERR_OPENCL_PROGRAM_CREATE	Error occurred when compiling an OpenCL program	GetLastError
ERR_OPENCL_QUEUE_CREATE	Failed to create a run queue in OpenCL	GetLastError
ERR_OPENCL_SET_KERNEL_PARAMETER	Error occurred when setting parameters for the Open	GetLastError

Constant	Description	Usage
	CL kernel	
ERR_OPENCL_TOO_LONG_KERNEL_NAME	Too long kernel name (OpenCL kernel)	GetLastError
ERR_OPENCL_WRONG_BUFFER_OFFSET	Invalid offset in the OpenCL buffer	GetLastError
ERR_OPENCL_WRONG_BUFFER_SIZE	Invalid size of the OpenCL buffer	GetLastError
ERR_PLAY_SOUND_FAILED	Sound playing failed	GetLastError
ERR_RESOURCE_NAME_DUPLICATED	The names of the dynamic and the static resource match	GetLastError
ERR_RESOURCE_NAME_IS_TOO_LONG	The resource name excee	GetLastError

Constant	Description	Usage
	ds 63 characters	
ERR_RESOURCE_NOT_FOUND	Resource with this name has not been found in EX5	GetLastError
ERR_RESOURCE_UNSUPPORTED_TYPE	Unsupported resource type or its size exceeds 16 Mb	GetLastError
ERR_SERIES_ARRAY	Timeseries cannot be used	GetLastError
ERR_SHORT_ARRAY_ONLY	Must be an array of type short	GetLastError
ERR_SMALL_ARRAY	Too small array, the starting position is outsid	GetLastError

Constant	Description	Usage
	e the array	
ERR_SMALL_ASSERIES_ARRAY	The receiving array is declared as AS_SERIES, and it is of insufficient size	GetLastError
ERR_STRING_OUT_OF_MEMORY	Not enough memory for the string	GetLastError
ERR_STRING_RESIZE_ERROR	Not enough memory for the relocation of string	GetLastError
ERR_STRING_SMALL_LEN	The string length is less than expected	GetLastError
ERR_STRING_TIME_ERROR	Error converting string	GetLastError

Constant	Description	Usage
	to date	
ERR_STRING_TOO_BIGNUMBER	Too large number, more than ULONG_MAX	GetLastError
ERR_STRING_UNKNOWNTYPE	Unknown data type when converting to a string	GetLastError
ERR_STRING_ZEROADDED	0 added to the string end, a useless operation	GetLastError
ERR_STRINGPOS_OUTOFRANGE	Position outside the string	GetLastError
ERR_STRUCT_WITHOBJECTS_ORCLASS	The structure contains objects of strings and/or	GetLastError

Constant	Description	Usage
	dynamic arrays and/or structure of such objects and/or classes	
ERR_SUCCESS	The operation completed successfully	GetLastError
ERR_TERMINAL_WRONG_PROPERTY	Wrong identifier of the terminal property	GetLastError
ERR_TOO_LONG_FILENAME	Too long file name	GetLastError
ERR_TOO_MANY_FILES	More than 64 files cannot be opened at the same time	GetLastError

Constant	Description	Usage
ERR_TOO_MANY_FORMATTERS	Amount of format specifiers more than the parameters	GetLastError
ERR_TOO_MANY_PARAMETERS	Amount of parameters more than the format specifiers	GetLastError
ERR_TRADE_DEAL_NOT_FOUND	Deal not found	GetLastError
ERR_TRADE_DISABLED	Trading by Expert Advisors prohibited	GetLastError
ERR_TRADE_ORDER_NOT_FOUND	Order not found	GetLastError
ERR_TRADE_POSITION_NOT_FOUND	Position not found	GetLastError
ERR_TRADE_SEND_FAILED	Trade request sendi	GetLastError

Constant	Description	Usage
	ng failed	
ERR_TRADE_WRONG_PROPERTY	Wrong trade property ID	GetLastError
ERR_USER_ERROR_FIRST	User defined errors start with this code	GetLastError
ERR_WEBREQUEST_CONNECT_FAILED	Failed to connect to specified URL	GetLastError
ERR_WEBREQUEST_INVALID_ADDRESS	Invalid URL	GetLastError
ERR_WEBREQUEST_REQUEST_FAILED	HTTP request failed	GetLastError
ERR_WEBREQUEST_TIMEOUT	Timeout exceeded	GetLastError
ERR_WRONG_DIRECTORYNAME	Wrong directory name	GetLastError
ERR_WRONG_FILEHANDLE	Wrong file handle	GetLastError

Constant	Description	Usage
ERR_WRONG_FILENAME	Invalid file name	GetLastError
ERR_WRONG_FORMATSTRING	Invalid format string	GetLastError
ERR_WRONG_INTERNAL_PARAMETER	Wrong parameter in the inner call of the client terminal function	GetLastError
ERR_WRONG_STRING_DATE	Wrong date in the string	GetLastError
ERR_WRONG_STRING_OBJECT	Damaged string object	GetLastError
ERR_WRONG_STRING_PARAMETER	Damaged parameter of string type	GetLastError
ERR_WRONG_STRING_TIME	Wrong time in the string	GetLastError
ERR_ZEROSIZE_ARRAY	An array	GetLastError

Constant	Description	Usage
	of zero length	
FILE_ACCESS_DATE	Date of the last access to the file	FileGetInteger
FILE_ANSI	Strings of ANSI type (one byte symbols). Flag is used in FileOpen().	FileOpen
FILE_BIN	Binary read/write mode (without string to string conversion). Flag is used in FileOpen().	FileOpen
FILE_COMMON	The file path in the	FileOpen, FileCopy, FileMove, FileIsExist

Constant	Description	Usage
	common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen(), FileCopy(), FileMove() and in FileIsExist() functions.	
FILE_CREATE_DATE	Date of creation	FileGetInteger
FILE_CSV	CSV file (all its elements are converted to strings of the appropriate	FileOpen

Constant	Description	Usage
	type, Unicode or ANSI, and separated by separator). Flag is used in FileOpen().	
FILE_END	Get the end of file sign	FileGetInteger
FILE_EXISTS	Check the existence	FileGetInteger
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)	FileGetInteger
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)	FileGetInteger
FILE_IS_COMMON	The file is opene	FileGetInteger

Constant	Description	Usage
	d in a shared folder of all terminals (see FILE_COMMON)	
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)	FileGetInteger
FILE_IS_READABLE	The opened file is readable (see FILE_READ)	FileGetInteger
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)	FileGetInteger
FILE_IS_WRITABLE	The opened file is writable (see FILE_	FileGetInteger

Constant	Description	Usage
	WRITE)	
FILE_LINE_END	Get the end of line sign	FileGetInteger
FILE_MODIFY_DATE	Date of the last modification	FileGetInteger
FILE_POSITION	Position of a pointer in the file	FileGetInteger
FILE_READ	File is opened for reading. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ	FileOpen

Constant	Description	Usage
	is required.	
FILE_REWRITE	Possibility for the file rewrite using functions FileCopy() and FileMove(). The file should exist or should be opened for writing, otherwise the file will not be opened.	FileCopy, FileMove
FILE_SHARE_READ	Shared access for reading from several	FileOpen

Constant	Description	Usage
	programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	
FILE_SHARE_WRITE	Shared access for writing from several programs. Flag is used in FileOpen(), but it	FileOpen

Constant	Description	Usage
	does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	
FILE_SIZE	File size in bytes	FileGetInteger
FILE_TXT	Simple text file (the same as csv file, but without taking into account the separators) . Flag is used in FileOpen().	FileOpen

Constant	Description	Usage
FILE_UNICODE	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen().	FileOpen
FILE_WRITE	File is opened for writing. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.	FileOpen
FLT_DIG	Number of signifi	Numerical Type Constants

Constant	Description	Usage
	cant decimal digits for float type	
FLT_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for float type)	Numerical Type Constants
FLT_MANT_DIG	Bits count in a mantissa for float type	Numerical Type Constants
FLT_MAX	Maximal value, which can be represented by float type	Numerical Type Constants
FLT_MAX_10_EXP	Maximal decimal	Numerical Type Constants

Constant	Description	Usage
	value of exponent degree for float type	
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	Numerical Type Constants
FLT_MIN	Minimal positive value, which can be represented by float type	Numerical Type Constants
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	Numerical Type Constants

Constant	Description	Usage
FLT_MIN_EXP	Minimal binary value of exponent degree for float type	Numerical Type Constants
FRIDAY	Friday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
GANN_DOWN_TREND	Line corresponding to the downward trend	ObjectSetInteger, ObjectGetInteger
GANN_UP_TREND	Line corresponding to the uptrend line	ObjectSetInteger, ObjectGetInteger
GATORJAW_LINE	Jaw line	Indicators Lines
GATORLIPS_LINE	Lips line	Indicators Lines
GATORTEETH_LINE	Teeth line	Indicators Lines
IDABORT	"Abort" button has been pressed	MessageBox

Constant	Description	Usage
IDCANCEL	"Cancel" button has been pressed	MessageBox
IDCONTINUE	"Continue" button has been pressed	MessageBox
IDIGNORE	"Ignore" button has been pressed	MessageBox
IDNO	"No" button has been pressed	MessageBox
IDOK	"OK" button has been pressed	MessageBox
IDRETRY	"Retry " button has been pressed	MessageBox
IDTRYAGAIN	"Try Again " butto	MessageBox

Constant	Description	Usage
	n has been pressed	
IDYES	"Yes" button has been pressed	MessageBox
IND_AC	Accelerator Oscillator	IndicatorCreate, IndicatorParameters
IND_AD	Accumulation/Distribution	IndicatorCreate, IndicatorParameters
IND_ADX	Average Directional Index	IndicatorCreate, IndicatorParameters
IND_ADXW	ADX by Welles Wilder	IndicatorCreate, IndicatorParameters
IND_ALLIGATOR	Alligator	IndicatorCreate, IndicatorParameters
IND_AMA	Adaptive Moving Average	IndicatorCreate, IndicatorParameters
IND_AO	Awesome Oscillator	IndicatorCreate, IndicatorParameters

Constant	Description	Usage
IND_ATR	Average True Range	IndicatorCreate, IndicatorParameters
IND_BANDS	Bollinger Bands®	IndicatorCreate, IndicatorParameters
IND_BEARS	Bears Power	IndicatorCreate, IndicatorParameters
IND_BULLS	Bulls Power	IndicatorCreate, IndicatorParameters
IND_BWMFI	Market Facilitation Index	IndicatorCreate, IndicatorParameters
IND_CCI	Commodity Channel Index	IndicatorCreate, IndicatorParameters
IND_CHAIKIN	Chaikin Oscillator	IndicatorCreate, IndicatorParameters
IND_CUSTOM	Custom indicator	IndicatorCreate, IndicatorParameters
IND_DEMA	Double Exponential Moving Average	IndicatorCreate, IndicatorParameters
IND_DEMARKER	DeMarker	IndicatorCreate, IndicatorParameters

Constant	Description	Usage
IND_ENVELOPES	Envelopes	IndicatorCreate, IndicatorParameters
IND_FORCE	Force Index	IndicatorCreate, IndicatorParameters
IND_FRACTALS	Fractals	IndicatorCreate, IndicatorParameters
IND_FRAMA	Fractal Adaptive Moving Average	IndicatorCreate, IndicatorParameters
IND_GATOR	Gator Oscillator	IndicatorCreate, IndicatorParameters
IND_ICHIMOKU	Ichimoku Kinko Hyo	IndicatorCreate, IndicatorParameters
IND_MA	Moving Average	IndicatorCreate, IndicatorParameters
IND_MACD	MACD	IndicatorCreate, IndicatorParameters
IND_MFI	Money Flow Index	IndicatorCreate, IndicatorParameters
IND_MOMENTUM	Momentum	IndicatorCreate, IndicatorParameters
IND_OBV	On Balance Volume	IndicatorCreate, IndicatorParameters
IND_OSMA	OsMA	IndicatorCreate, IndicatorParameters
IND_RSI	Relative Stren	IndicatorCreate, IndicatorParameters

Constant	Description	Usage
	gth Index	
IND_RVI	Relative Vigor Index	IndicatorCreate, IndicatorParameters
IND_SAR	Parabolic SAR	IndicatorCreate, IndicatorParameters
IND_STDDEV	Standard Deviation	IndicatorCreate, IndicatorParameters
IND_STOCHASTIC	Stochastic Oscillator	IndicatorCreate, IndicatorParameters
IND_TEMA	Triple Exponential Moving Average	IndicatorCreate, IndicatorParameters
IND_TRIX	Triple Exponential Moving Averages Oscillator	IndicatorCreate, IndicatorParameters
IND_VIDYA	Variable Index Dynamic Average	IndicatorCreate, IndicatorParameters
IND_VOLUMES	Volumes	IndicatorCreate, IndicatorParameters

Constant	Description	Usage
IND_WPR	Williams' Percent Range	IndicatorCreate, IndicatorParameters
INDICATOR_CALCULATIONS	Auxiliary buffers for intermediate calculations	SetIndexBuffer
INDICATOR_COLOR_INDEX	Color	SetIndexBuffer
INDICATOR_DATA	Data to draw	SetIndexBuffer
INDICATOR_DIGITS	Accuracy of drawing of indicator values	IndicatorSetInteger
INDICATOR_HEIGHT	Fixed height of the indicator's window (the preprocessor command #property indicator_height)	IndicatorSetInteger
INDICATOR_LEVELCOLOR	Color of the	IndicatorSetInteger

Constant	Description	Usage
	level line	
INDICATOR_LEVELS	Number of levels in the indicator window	IndicatorSetInteger
INDICATOR_LEVELSTYLE	Style of the level line	IndicatorSetInteger
INDICATOR_LEVELTEXT	Level description	IndicatorSetString
INDICATOR_LEVELVALUE	Level value	IndicatorSetDouble
INDICATOR_LEVELWIDTH	Thickness of the level line	IndicatorSetInteger
INDICATOR_MAXIMUM	Maximum of the indicator window	IndicatorSetDouble
INDICATOR_MINIMUM	Minimum of the indicator window	IndicatorSetDouble
INDICATOR_SHORTNAME	Short indicator name	IndicatorSetString

Constant	Description	Usage
INT_MAX	Maximal value, which can be represented by int type	Numerical Type Constants
INT_MIN	Minimal value, which can be represented by int type	Numerical Type Constants
INVALID_HANDLE	Incorrect handle	Other Constants
IS_DEBUG_MODE	Flag that a mq5-program operates in debug mode	Other Constants
IS_PROFILE_MODE	Flag that a mq5-program operates in profiling mode	Other Constants

Constant	Description	Usage
KIJUNSEN_LINE	Kijun-sen line	Indicators Lines
LICENSE_DEMO	A trial version of a paid product from the Market. It works only in the strategy tester	MQLInfoInteger
LICENSE_FREE	A free unlimited version	MQLInfoInteger
LICENSE_FULL	A purchased licensed version allows at least 5 activations. The number of activations is specified by seller.	MQLInfoInteger

Constant	Description	Usage
	Seller may increase the allowed number of activations	
LICENSE_TIME	A version with a limited term license	MQLInfoInteger
LONG_MAX	Maximal value, which can be represented by long type	Numerical Type Constants
LONG_MIN	Minimal value, which can be represented by long type	Numerical Type Constants
LOWER_BAND	Lower limit	Indicators Lines

Constant	Description	Usage
LOWER_HISTOGRAM	Bottom histogram	Indicators Lines
LOWER_LINE	Bottom line	Indicators Lines
M_1_PI	1/pi	Mathematical Constants
M_2_PI	2/pi	Mathematical Constants
M_2_SQRTPI	2/sqrt(pi)	Mathematical Constants
M_E	e	Mathematical Constants
M_LN10	ln(10)	Mathematical Constants
M_LN2	ln(2)	Mathematical Constants
M_LOG10E	log10(e)	Mathematical Constants
M_LOG2E	log2(e)	Mathematical Constants
M_PI	pi	Mathematical Constants
M_PI_2	pi/2	Mathematical Constants
M_PI_4	pi/4	Mathematical Constants
M_SQRT1_2	1/sqrt(2)	Mathematical Constants
M_SQRT2	sqrt(2)	Mathematical Constants
MAIN_LINE	Main line	Indicators Lines
MB_ABORTRETRYIGNORE	Message window contains three buttons: Abort, Retry	MessageBox

Constant	Description	Usage
	and Ignore	
MB_CANCELTRYCONTINUE	Message window contains three buttons: Cancel, Try Again, Continue	MessageBox
MB_DEFBUTTON1	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified	MessageBox

Constant	Description	Usage
MB_DEFBUTTON2	The second button is default	MessageBox
MB_DEFBUTTON3	The third button is default	MessageBox
MB_DEFBUTTON4	The fourth button is default	MessageBox
MB_ICONEXCLAMATION, MB_ICONWARNING	The exclamation/warning sign icon	MessageBox
MB_ICONINFORMATION, MB_ICONASTERISK	The encircled i sign	MessageBox
MB_ICONQUESTION	The question sign icon	MessageBox
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	The STOP sign icon	MessageBox
MB_OK	Message window	MessageBox

Constant	Description	Usage
	contains only one button: OK. Default	
MB_OKCANCEL	Message window contains two buttons: OK and Cancel	MessageBox
MB_RETRYCANCEL	Message window contains two buttons: Retry and Cancel	MessageBox
MB_YESNO	Message window contains two buttons: Yes and No	MessageBox

Constant	Description	Usage
MB_YESNOCANCEL	Message window contains three buttons: Yes, No and Cancel	MessageBox
MINUSDI_LINE	Line – DI	Indicators Lines
MODE_EMA	Exponential averaging	Smoothing Methods
MODE_LWMA	Linear-weighted averaging	Smoothing Methods
MODE_SMA	Simple averaging	Smoothing Methods
MODE_SMMA	Smoothed averaging	Smoothing Methods
MONDAY	Monday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
MQL_DEBUG	The flag, that indicates the	MQLInfoInteger

Constant	Description	Usage
	debug mode	
MQL_DLLS_ALLOWED	The permission to use DLL for the given executed program	MQLInfoInteger
MQL_FRAME_MODE	The flag, that indicates the Expert Advisor operating in gathering optimization result frames mode	MQLInfoInteger
MQL_LICENSE_TYPE	Type of license of the EX5 module. The licens	MQLInfoInteger

Constant	Description	Usage
	e refers to the EX5 module, from which a request is made using MQLInfoInteger(MQL_LICENSE_TYPE).	
MQL_MEMORY_LIMIT	Maximum possible amount of dynamic memory for MQL5 program in MB	[MQLInfoInteger]()
MQL_MEMORY_USED	The memory size used by MQL5 program in MB	[MQLInfoInteger]()
MQL_OPTIMIZATION	The flag,	[MQLInfoInteger]()

Constant	Description	Usage
	that indicates the optimization process	
MQL_PROFILER	The flag, that indicates the program operating in the code profiling mode	MQLInfoInteger
MQL_PROGRAM_NAME	Name of the mql5-program executed	MQLInfoString
MQL_PROGRAM_PATH	Path for the given executed program	MQLInfoString
MQL_PROGRAM_TYPE	Type of the mql5 program	MQLInfoInteger

Constant	Description	Usage
MQL_SIGNALS_ALLOWED	The permission to modify the Signals for the given executed program	MQLInfoInteger
MQL_TESTER	The flag, that indicates the tester process	MQLInfoInteger
MQL_TRADE_ALLOWED	The permission to trade for the given executed program	MQLInfoInteger
MQL_VISUAL_MODE	The flag, that indicates the visual tester process	MQLInfoInteger

Constant	Description	Usage
NULL	Zero for any types	Other Constants
OBJ_ALL_PERIODS	The object is drawn in all timeframes	ObjectSetInteger, ObjectGetInteger
OBJ_ARROW	Arrow	Object Types
OBJ_ARROW_BUY	Buy Sign	Object Types
OBJ_ARROW_CHECK	Check Sign	Object Types
OBJ_ARROW_DOWN	Arrow Down	Object Types
OBJ_ARROW_LEFT_PRICE	Left Price Label	Object Types
OBJ_ARROW_RIGHT_PRICE	Right Price Label	Object Types
OBJ_ARROW_SELL	Sell Sign	Object Types
OBJ_ARROW_STOP	Stop Sign	Object Types
OBJ_ARROW_THUMB_DOWN	Thumbs Down	Object Types
OBJ_ARROW_THUMB_UP	Thumbs Up	Object Types
OBJ_ARROW_UP	Arrow Up	Object Types
OBJ_ARROWED_LINE	Arrowed Line	Object Types

Constant	Description	Usage
OBJ_BITMAP	Bitmap	Object Types
OBJ_BITMAP_LABEL	Bitmap Label	Object Types
OBJ_BUTTON	Button	Object Types
OBJ_CHANNEL	Equidistant Channel	Object Types
OBJ_CHART	Chart	Object Types
OBJ_CYCLES	Cycle Lines	Object Types
OBJ_EDIT	Edit	Object Types
OBJ_ELLIOTWAVE3	Elliott Correction Wave	Object Types
OBJ_ELLIOTWAVE5	Elliott Motive Wave	Object Types
OBJ_ELLIPSE	Ellipse	Object Types
OBJ_EVENT	The "Event" object corresponding to an event in the economic calendar	Object Types
OBJ_EXPANSION	Fibonacci	Object Types

Constant	Description	Usage
	Expansion	
OBJ_FIBO	Fibonacci Retracement	Object Types
OBJ_FIBOARC	Fibonacci Arcs	Object Types
OBJ_FIBOCHANNEL	Fibonacci Channel	Object Types
OBJ_FIBOFAN	Fibonacci Fan	Object Types
OBJ_FIBOTIMES	Fibonacci Time Zones	Object Types
OBJ_GANNFAN	Gann Fan	Object Types
OBJ_GANNGRID	Gann Grid	Object Types
OBJ_GANNLINE	Gann Line	Object Types
OBJ_HLINE	Horizontal Line	Object Types
OBJ_LABEL	Label	Object Types
OBJ_NO_PERIODS	The object is not drawn in all timeframes	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_D1	The object	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	is drawn in day charts	
OBJ_PERIOD_H1	The object is drawn in 1-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_H12	The object is drawn in 12-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_H2	The object is drawn in 2-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_H3	The object is drawn in 3-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_H4	The object is drawn in 4-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_H6	The object is drawn in 6-	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	hour chart	
OBJ_PERIOD_H8	The object is drawn in 8-hour chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M1	The object is drawn in 1-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M10	The object is drawn in 10-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M12	The object is drawn in 12-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M15	The object is drawn in 15-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M2	The object is	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	drawn in 2-minute chart	
OBJ_PERIOD_M20	The object is drawn in 20-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M3	The object is drawn in 3-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M30	The object is drawn in 30-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M4	The object is drawn in 4-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_M5	The object is drawn in 5-minute chart	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
OBJ_PERIOD_M6	The object is drawn in 6-minute chart	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_MN1	The object is drawn in month charts	ObjectSetInteger, ObjectGetInteger
OBJ_PERIOD_W1	The object is drawn in week charts	ObjectSetInteger, ObjectGetInteger
OBJ_PITCHFORK	Andrews’ Pitchfork	Object Types
OBJ_RECTANGLE	Rectangle	Object Types
OBJ_RECTANGLE_LABEL	The "Rectangle label" object for creating and designing the custom graphical	Object Types

Constant	Description	Usage
	interface.	
OBJ_REGRESSION	Linear Regression Channel	Object Types
OBJ_STDDEVCHANNEL	Standard Deviation Channel	Object Types
OBJ_TEXT	Text	Object Types
OBJ_TREND	Trend Line	Object Types
OBJ_TRENDBYANGLE	Trend Line By Angle	Object Types
OBJ_TRIANGLE	Triangle	Object Types
OBJ_VLINE	Vertical Line	Object Types
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	ObjectSetInteger, ObjectGetInteger
OBJPROP_ANCHOR	Location of the anchor point of a	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	graphical object	
OBJPROP_ANGLE	Angle. For the objects with no angle specified, created from a program, the value is equal to EMPTY_VALUE	ObjectSetDouble, ObjectGetDouble
OBJPROP_ARROWCODE	Arrow code for the Arrow object	ObjectSetInteger, ObjectGetInteger
OBJPROP_BACK	Object in the background	ObjectSetInteger, ObjectGetInteger
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_B	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	UTTON, OBJ_RECTANGLE_LABEL	
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also Resources	ObjectSetString, ObjectGetString
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	ObjectSetInteger, ObjectGetInteger
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	ObjectSetInteger, ObjectGetInteger
OBJPROP_CHART_ID	ID of the "Chart" object	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	(OBJ_CHART). It allows working with the properties of this object like with a normal chart using the functions described in Chart Operations, but there some exceptions.	
OBJPROP_CHART_SCALE	The scale for the Chart object	ObjectSetInteger, ObjectGetInteger
OBJPROP_COLOR	Color	ObjectSetInteger, ObjectGetInteger
OBJPROP_CORNER	The corner of the chart to link a graphi	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	cal object	
OBJPROP_CREATETIME	Time of object creation	ObjectSetInteger, ObjectGetInteger
OBJPROP_DATE_SCALE	Displaying the time scale for the Chart object	ObjectSetInteger, ObjectGetInteger
OBJPROP_DEGREE	Level of the Elliott Wave Marking	ObjectSetInteger, ObjectGetInteger
OBJPROP_DEVIATION	Deviation for the Standard Deviation Channel	ObjectSetDouble, ObjectGetDouble
OBJPROP_DIRECTION	Trend of the Gann object	ObjectSetInteger, ObjectGetInteger
OBJPROP_DRAWLINES	Displaying lines for marking the	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	Elliott Wave	
OBJPROP_ELLIPSE	Showing the full ellipse of the Fibonacci Arc object (OBJ_FIBOARC)	ObjectSetInteger, ObjectGetInteger
OBJPROP_FILL	Fill an object with color (for OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION)	ObjectSetInteger, ObjectGetInteger
OBJPROP_FONT	Font	ObjectSetString, ObjectGetString
OBJPROP_FONTSIZE	Font size	ObjectSetInteger, ObjectGetInteger
OBJPROP_HIDDEN	Prohibit showi	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	ng of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects". The true value allows to hide an object from the list. By default, true is set to the objects that display calendar event	

Constant	Description	Usage
	s, trading history and to the objects created from MQL5 programs. To see such graphical objects and access their properties, click on the "All" button in the "List of objects" window.	
OBJPROP_LEVELCOLOR	Color of the line-level	ObjectSetInteger, ObjectGetInteger
OBJPROP_LEVELS	Number of levels	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
OBJPROP_LEVELSTYLE	Style of the line-level	ObjectSetInteger, ObjectGetInteger
OBJPROP_LEVELTEXT	Level description	ObjectSetString, ObjectGetString
OBJPROP_LEVELVALUE	Level value	ObjectSetDouble, ObjectGetDouble
OBJPROP_LEVELWIDTH	Thickness of the line-level	ObjectSetInteger, ObjectGetInteger
OBJPROP_NAME	Object name	ObjectSetString, ObjectGetString
OBJPROP_PERIOD	Timeframe for the Chart object	ObjectSetInteger, ObjectGetInteger
OBJPROP_PRICE	Price coordinate	ObjectSetDouble, ObjectGetDouble
OBJPROP_PRICE_SCALE	Displaying the price scale for the Chart object	ObjectSetInteger, ObjectGetInteger
OBJPROP_RAY	A vertical line goes through all	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	the windows of a chart	
OBJPROP_RAY_LEFT	Ray goes to the left	ObjectSetInteger, ObjectGetInteger
OBJPROP_RAY_RIGHT	Ray goes to the right	ObjectSetInteger, ObjectGetInteger
OBJPROP_READONLY	Ability to edit text in the Edit object	ObjectSetInteger, ObjectGetInteger
OBJPROP_SCALE	Scale (properties of Gann objects and Fibonacci Arcs)	ObjectSetDouble, ObjectGetDouble
OBJPROP_SELECTABLE	Object availability	ObjectSetInteger, ObjectGetInteger
OBJPROP_SELECTED	Object is selected	ObjectSetInteger, ObjectGetInteger
OBJPROP_STATE	Button state (pressed /	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	depressed)	
OBJPROP_STYLE	Style	ObjectSetInteger, ObjectGetInteger
OBJPROP_SYMBOL	Symbol for the Chart object	ObjectSetString, ObjectGetString
OBJPROP_TEXT	Description of the object (the text contained in the object)	ObjectSetString, ObjectGetString
OBJPROP_TIME	Time coordinate	ObjectSetInteger, ObjectGetInteger
OBJPROP_TIMEFRAMES	Visibility of an object at timeframes	ObjectSetInteger, ObjectGetInteger
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip gener	ObjectSetString, ObjectGetString

Constant	Description	Usage
	ated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value to it	
OBJPROP_TYPE	Object type	ObjectSetInteger, ObjectGetInteger
OBJPROP_WIDTH	Line thickness	ObjectSetInteger, ObjectGetInteger
OBJPROP_XDISTANCE	The distance in pixels along the X axis from the binding corner (see note)	ObjectSetInteger, ObjectGetInteger
OBJPROP_XOFFSET	The X coordinate of the upper	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	
OBJPROP_XSIZE	The object's width along the X axis	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	
OBJPROP_YDISTANCE	The distance in pixels along the Y axis from the binding corner (see note)	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
OBJPROP_YOFFSET	The Y coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	ObjectSetInteger, ObjectGetInteger
OBJPROP_YSIZE	The object	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	
OBJPROP_ZORDER	Priority of a graphical object for receiving events of	ObjectSetInteger, ObjectGetInteger

Constant	Description	Usage
	clicking on a chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When applying objects one over another, only one of them with the highest priority will receive the CHARTEVENT_CLICK	

Constant	Description	Usage
	event.	
ORDER_COMMENT	Order comment	OrderGetString, HistoryOrderGetString
ORDER_FILLING_FOK	This filling policy means that an order can be filled only in the specified amount. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The requir	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
	ed volume can be filled using several offers available on the market at the moment.	
ORDER_FILLING_IOC	This mode means that a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
	the the entire volume of an order cannot be filled, the available volume of it will be filled, and the remaining volume will be canceled.	
ORDER_FILLING_RETURN	This policy is used only for market orders (ORDER_TYPE_BUY and ORDER_TYPE_SELL), limit and stop limit	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
	orders (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT) and only for the symbols with Market or Exchange execution. In case of partial filling a market or limit order with	

Constant	Description	Usage
	remaining volume is not canceled but processed further. For the activation of the ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT orders, a corresponding limit order ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT with the ORDER_FIL	

Constant	Description	Usage
	LING_RETURN execution type is created.	
ORDER_MAGIC	ID of an Expert Advisor that has placed the order (designed to ensure that each Expert Advisor places its own unique number)	OrderGetInteger, HistoryOrderGetInteger
ORDER_POSITION_ID	Position identifier that is set to an order as soon	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
	as it is executed. Each executed order results in a deal that opens or modifies an already existing position. The identifier of exactly this position is set to the executed order at this moment.	
ORDER_PRICE_CURRENT	The current price of the order symbol	OrderGetDouble, HistoryOrderGetDouble

Constant	Description	Usage
ORDER_PRICE_OPEN	Price specified in the order	OrderGetDouble, HistoryOrderGetDouble
ORDER_PRICE_STOPLIMIT	The Limit order price for the StopLimit order	OrderGetDouble, HistoryOrderGetDouble
ORDER_SL	Stop Loss value	OrderGetDouble, HistoryOrderGetDouble
ORDER_STATE	Order state	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_CANCELED	Order canceled by client	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_EXPIRED	Order expired	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_FILLED	Order fully executed	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_PARTIAL	Order partially executed	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_PLACED	Order accepted	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_REJECTED	Order rejected	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
ORDER_STATE_REQUEST_ADD	Order is being registered (placing to the trading system)	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_REQUEST_CANCEL	Order is being deleted (deleting from the trading system)	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_REQUEST_MODIFY	Order is being modified (changing its parameters)	OrderGetInteger, HistoryOrderGetInteger
ORDER_STATE_STARTED	Order checked, but not yet accepted by broker	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
ORDER_SYMBOL	Symbol of the order	OrderGetString, HistoryOrderGetString
ORDER_TIME_DAY	Good till current trade day order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_DONE	Order execution or cancellation time	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_DONE_MSC	Order execution/cancellation time in milliseconds since 01.01.1970	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_EXPIRATION	Order expiration time	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_GTC	Good till cancel order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_SETUP	Order setup time	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
ORDER_TIME_SETUP_MSC	The time of placing an order for execution in milliseconds since 01.01.1970	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_SPECIFIED	Good till expired order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TIME_SPECIFIED_DAY	The order will be effective till 23:59:59 of the specified day. If this time is outside a trading session, the order expires in the neare	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
	st trading time.	
ORDER_TP	Take Profit value	OrderGetDouble, HistoryOrderGetDouble
ORDER_TYPE	Order type	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_BUY	Market Buy order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_BUY_LIMIT	Buy Limit pending order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_BUY_STOP	Buy Stop pending order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_BUY_STOP_LIMIT	Upon reaching the order price, a pending Buy Limit order is placed at the StopLimit price	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_FILLING	Order filling type	OrderGetInteger, HistoryOrderGetInteger

Constant	Description	Usage
ORDER_TYPE_SELL	Market Sell order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_SELL_LIMIT	Sell Limit pending order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_SELL_STOP	Sell Stop pending order	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_SELL_STOP_LIMIT	Upon reaching the order price, a pending Sell Limit order is placed at the StopLimit price	OrderGetInteger, HistoryOrderGetInteger
ORDER_TYPE_TIME	Order lifetime	OrderGetInteger, HistoryOrderGetInteger
ORDER_VOLUME_CURRENT	Order current volume	OrderGetDouble, HistoryOrderGetDouble
ORDER_VOLUME_INITIAL	Order initial volume	OrderGetDouble, HistoryOrderGetDouble

Constant	Description	Usage
PERIOD_CURRENT	Current timeframe	Chart Timeframes
PERIOD_D1	1 day	Chart Timeframes
PERIOD_H1	1 hour	Chart Timeframes
PERIOD_H12	12 hours	Chart Timeframes
PERIOD_H2	2 hours	Chart Timeframes
PERIOD_H3	3 hours	Chart Timeframes
PERIOD_H4	4 hours	Chart Timeframes
PERIOD_H6	6 hours	Chart Timeframes
PERIOD_H8	8 hours	Chart Timeframes
PERIOD_M1	1 minute	Chart Timeframes
PERIOD_M10	10 minutes	Chart Timeframes
PERIOD_M12	12 minutes	Chart Timeframes
PERIOD_M15	15 minutes	Chart Timeframes
PERIOD_M2	2 minutes	Chart Timeframes
PERIOD_M20	20 minutes	Chart Timeframes
PERIOD_M3	3 minut	Chart Timeframes

Constant	Description	Usage
	es	
PERIOD_M30	30 minutes	Chart Timeframes
PERIOD_M4	4 minutes	Chart Timeframes
PERIOD_M5	5 minutes	Chart Timeframes
PERIOD_M6	6 minutes	Chart Timeframes
PERIOD_MN1	1 month	Chart Timeframes
PERIOD_W1	1 week	Chart Timeframes
PLOT_ARROW	Arrow code for style DRAW_ARROW	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_ARROW_SHIFT	Vertical shift of arrows for style DRAW_ARROW	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_COLOR_INDEXES	The number of colors	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_DRAW_BEGIN	Number of initial	PlotIndexSetInteger, PlotIndexGetInteger

Constant	Description	Usage
	bars without drawing and values in the Data Window	
PLOT_DRAW_TYPE	Type of graphical construction	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_EMPTY_VALUE	An empty value for plotting, for which there is no drawing	PlotIndexSetDouble
PLOT_LABEL	The name of the indicator graphical series to display in the Data Window. When worki	PlotIndexSetString

Constant	Description	Usage
	ng with complex graphical styles requiring several indicator buffers for display, the names for each buffer can be specified using ";" as a separator. Sample code is shown in DRAW_CANDLES	
PLOT_LINE_COLOR	The index of a buffer containing the drawi	PlotIndexSetInteger, PlotIndexGetInteger

Constant	Description	Usage
	ng color	
PLOT_LINE_STYLE	Drawing line style	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_LINE_WIDTH	The thickness of the drawing line	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_SHIFT	Shift of indicator plotting along the time axis in bars	PlotIndexSetInteger, PlotIndexGetInteger
PLOT_SHOW_DATA	Sign of display of construction values in the Data Window	PlotIndexSetInteger, PlotIndexGetInteger
PLUSDI_LINE	Line +DI	Indicators Lines
POINTER_AUTOMATIC	Pointer of any objects create	CheckPointer

Constant	Description	Usage
	d automatically (not using new())	
POINTER_DYNAMIC	Pointer of the object created by the new() operator	CheckPointer
POINTER_INVALID	Incorrect pointer	CheckPointer
POSITION_COMMENT	Position comment	PositionGetString
POSITION_COMMISSION	Commission	PositionGetDouble
POSITION_IDENTIFIER	Position identifier is a unique number that is assigned to every newly opened positi	PositionGetInteger

Constant	Description	Usage
	on and doesn't change during the entire lifetime of the position. Position turnover doesn't change its identifier.	
POSITION_MAGIC	Position magic number (see ORDER_MAGIC)	PositionGetInteger
POSITION_PRICE_CURRENT	Current price of the position symbol	PositionGetDouble
POSITION_PRICE_OPEN	Position open price	PositionGetDouble

Constant	Description	Usage
POSITION_PROFIT	Current profit	PositionGetDouble
POSITION_SL	Stop Loss level of opened position	PositionGetDouble
POSITION_SWAP	Cumulative swap	PositionGetDouble
POSITION_SYMBOL	Symbol of the position	PositionGetString
POSITION_TIME	Position open time	PositionGetInteger
POSITION_TIME_MSC	Position opening time in milliseconds since 01.01.1970	PositionGetInteger
POSITION_TIME_UPDATE	Position changing time in seconds since	PositionGetInteger

Constant	Description	Usage
	01.01.1970	
POSITION_TIME_UPDATE_MSC	Position changing time in milliseconds since 01.01.1970	PositionGetInteger
POSITION_TP	Take Profit level of opened position	PositionGetDouble
POSITION_TYPE	Position type	PositionGetInteger
POSITION_TYPE_BUY	Buy	PositionGetInteger
POSITION_TYPE_SELL	Sell	PositionGetInteger
POSITION_VOLUME	Position volume	PositionGetDouble
PRICE_CLOSE	Close price	Price Constants
PRICE_HIGH	The maximum price for the period	Price Constants
PRICE_LOW	The minim	Price Constants

Constant	Description	Usage
	um price for the period	
PRICE_MEDIAN	Median price, (high + low)/2	Price Constants
PRICE_OPEN	Open price	Price Constants
PRICE_TYPICAL	Typical price, (high + low + close)/3	Price Constants
PRICE_WEIGHTED	Average price, (high + low + close + close)/4	Price Constants
PROGRAM_EXPERT	Expert	MQLInfoInteger
PROGRAM_INDICATOR	Indicator	MQLInfoInteger
PROGRAM_SCRIPT	Script	MQLInfoInteger
REASON_ACCOUNT	Another account has been activa	UninitializeReason, OnDeinit

Constant	Description	Usage
	ted or reconnection to the trade server has occurred due to changes in the account settings	
REASON_CHARTCHANGE	Symbol or chart period has been changed	UninitializeReason, OnDeinit
REASON_CHARTCLOSE	Chart has been closed	UninitializeReason, OnDeinit
REASON_CLOSE	Terminal has been closed	UninitializeReason, OnDeinit
REASON_INITFAILED	This value means that OnInit() handler has returned a	UninitializeReason, OnDeinit

Constant	Description	Usage
	nonzero value	
REASON_PARAMETERS	Input parameters have been changed by a user	UninitializeReason, OnDeinit
REASON_PROGRAM	Expert Advisor terminated its operation by calling the ExpertRemove() function	UninitializeReason, OnDeinit
REASON_RECOMPILE	Program has been recompiled	UninitializeReason, OnDeinit
REASON_REMOVE	Program has been deleted from the chart	UninitializeReason, OnDeinit

Constant	Description	Usage
REASON_TEMPLATE	A new template has been applied	UninitializeReason, OnDeinit
SATURDAY	Saturday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
SEEK_CUR	Current position of a file pointer	FileSeek
SEEK_END	File end	FileSeek
SEEK_SET	File beginning	FileSeek
SENKOUSPANA_LINE	Senkou Span A line	Indicators Lines
SENKOUSPANB_LINE	Senkou Span B line	Indicators Lines
SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	SeriesInfoInteger

Constant	Description	Usage
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	SeriesInfoInteger
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	SeriesInfoInteger
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	SeriesInfoInteger
SERIES_SYNCHRONIZED	Symbol/period data synchr	SeriesInfoInteger

Constant	Description	Usage
	onization flag for the current moment	
SERIES_TERMINAL_FIRSTDATE	The very first date in the history of the symbol in the client terminal, regardless of the timeframe	SeriesInfoInteger
SHORT_MAX	Maximal value, which can be represented by short type	Numerical Type Constants
SHORT_MIN	Minimal value, which can be repres	Numerical Type Constants

Constant	Description	Usage
	ented by short type	
SIGNAL_BASE_AUTHOR_LOGIN	Author login	SignalBaseGetString
SIGNAL_BASE_BALANCE	Account balance	SignalBaseGetDouble
SIGNAL_BASE_BROKER	Broker name (company)	SignalBaseGetString
SIGNAL_BASE_BROKER_SERVER	Broker server	SignalBaseGetString
SIGNAL_BASE_CURRENCY	Signal base currency	SignalBaseGetString
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)	SignalBaseGetInteger
SIGNAL_BASE_DATE_STARTED	Monitoring starting date	SignalBaseGetInteger
SIGNAL_BASE_EQUITY	Account	SignalBaseGetDouble

Constant	Description	Usage
	equity	
SIGNAL_BASE_GAIN	Account gain	SignalBaseGetDouble
SIGNAL_BASE_ID	Signal ID	SignalBaseGetInteger
SIGNAL_BASE_LEVERAGE	Account leverage	SignalBaseGetInteger
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown	SignalBaseGetDouble
SIGNAL_BASE_NAME	Signal name	SignalBaseGetString
SIGNAL_BASE_PIPS	Profit in pips	SignalBaseGetInteger
SIGNAL_BASE_PRICE	Signal subscription price	SignalBaseGetDouble
SIGNAL_BASE_RATING	Position in rating	SignalBaseGetInteger
SIGNAL_BASE_ROI	Return on Investment (%)	SignalBaseGetDouble
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers	SignalBaseGetInteger
SIGNAL_BASE_TRADE_MODE	Account type (0-	SignalBaseGetInteger

Constant	Description	Usage
	real, 1-demo, 2-contest)	
SIGNAL_BASE_TRADES	Number of trades	SignalBaseGetInteger
SIGNAL_INFO_CONFIRMATIONS_DISABLED	The flag enables synchronization without confirmation dialog	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_DEPOSIT_PERCENT	Deposit percent (%)	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_EQUITY_LIMIT	Equity limit	SignalInfoGetDouble, SignalInfoSetDouble
SIGNAL_INFO_ID	Signal id, r/o	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_NAME	Signal name, r/o	SignalInfoGetString
SIGNAL_INFO_SLIPPAGE	Slippage	SignalInfoGetDouble, SignalInfoSetDouble

Constant	Description	Usage
	(used when placing market orders in synchronization of positions and copying of trades)	
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o	SignalInfoGetInteger, SignalInfoSetInteger
SIGNAL_INFO_VOLUME_PERCENT	Maximum percent of deposit used	SignalInfoGetDouble, SignalInfoSetDouble

Constant	Description	Usage
	(%), r/o	
SIGNAL_LINE	Signal line	Indicators Lines
STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous drawdowns; here the largest value is taken	TesterStatistics
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at	TesterStatistics

Constant	Description	Usage
	the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the	TesterStatistics

Constant	Description	Usage
	relative drawdown value in percents is calculated. The greatest value is returned	
STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	TesterStatistics

Constant	Description	Usage
STAT_BALANCEMIN	Minimum balance value	TesterStatistics
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	TesterStatistics
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	TesterStatistics
STAT_CONPROFITMAX	Maximum profit in a	TesterStatistics

Constant	Description	Usage
	series of profitable trades. The value is greater than or equal to zero	
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	TesterStatistics
STAT_CUSTOM_ONTESTER	The value of the calculated custom optimizatio	TesterStatistics

Constant	Description	Usage
	n criterion returned by the OnTester() function	
STAT_DEALS	The number of deals	TesterStatistics
STAT_EQUITY_DD	Maximum equity drawdown in monetary terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	TesterStatistics

Constant	Description	Usage
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	TesterStatistics
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numer	TesterStatistics

Constant	Description	Usage
	ous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity drawdown in monetary terms	TesterStatistics

Constant	Description	Usage
	(STAT_EQUITY_DD).	
STAT_EQUITYMIN	Minimum equity value	TesterStatistics
STAT_EXPECTED_PAYOFF	Expected payoff	TesterStatistics
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	TesterStatistics
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal	TesterStatistics

Constant	Description	Usage
	to zero	
STAT_INITIAL_DEPOSIT	The value of the initial deposit	[TesterStatistics]()
STAT_LONG_TRADES	Long trades	[TesterStatistics]()
STAT_LOSS_TRADES	Losing trades	[TesterStatistics]()
STAT_LOSSTRADES_AVGCON	Average length of a losing series of trades	[TesterStatistics]()
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	[TesterStatistics]()
STAT_MAX_CONLOSSES	The total loss of the longest series	[TesterStatistics]()

Constant	Description	Usage
	of losing trades	
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	TesterStatistics
STAT_MAX_CONWINS	The total profit of the longest series of profitable trades	TesterStatistics
STAT_MAX_LOSSTRADE	Maximum loss – the lowest value of all losing trades. The value is less than or equal	TesterStatistics

Constant	Description	Usage
	to zero	
STAT_MAX_PROFITTRADE	Maximum profit – the largest value of all profitable trades. The value is greater than or equal to zero	TesterStatistics
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	TesterStatistics
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS	TesterStatistics

Constant	Description	Usage
	(STAT_GROSS_LOSS is always less than or equal to zero)	
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of STAT_GROSS_PROFIT/STAT_GROSS_LOSS. If STAT_GROSS_LOSS=0, the profit factor is equal to DBL_MAX	TesterStatistics
STAT_PROFIT_LONGTRADES	Profitable long trades	TesterStatistics
STAT_PROFIT_SHORTTRADES	Profitable	TesterStatistics

Constant	Description	Usage
	short trades	
STAT_PROFIT_TRADES	Profitable trades	TesterStatistics
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	TesterStatistics
STAT_RECOVERY_FACTOR	Recovery factor, equal to the ratio of STAT_PROFIT/STAT_BALANCE_DD	TesterStatistics
STAT_SHARPE_RATIO	Sharpe ratio	TesterStatistics
STAT_SHORT_TRADES	Short trades	TesterStatistics
STAT_TRADES	The number of trades	TesterStatistics
STAT_WITHDRAWAL	Money withdrawn from an	TesterStatistics

Constant	Description	Usage
	account	
STO_CLOSECLOSE	Calculation is based on Close/Close prices	Price Constants
STO_LOWHIGH	Calculation is based on Low/High prices	Price Constants
STYLE_DASH	Broken line	Drawing Styles
STYLE_DASHDOT	Dash-dot line	Drawing Styles
STYLE_DASHDOTDOT	Dash - two points	Drawing Styles
STYLE_DOT	Dotted line	Drawing Styles
STYLE_SOLID	Solid line	Drawing Styles
SUNDAY	Sunday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
SYMBOL_ASK	Ask - best buy offer	SymbolInfoDouble
SYMBOL_ASKHIGH	Maximal Ask of	SymbolInfoDouble

Constant	Description	Usage
	the day	
SYMBOL_ASKLOW	Minimal Ask of the day	SymbolInfoDouble
SYMBOL_BANK	Feeder of the current quote	SymbolInfoString
SYMBOL_BASIS	The underlying asset of a derivative	SymbolInfoString
SYMBOL_BID	Bid - best sell offer	SymbolInfoDouble
SYMBOL_BIDHIGH	Maximal Bid of the day	SymbolInfoDouble
SYMBOL_BIDLOW	Minimal Bid of the day	SymbolInfoDouble
SYMBOL_CALC_MODE_CFD	CFD mode - calculation of margin and profit for CFD	SymbolInfoInteger

Constant	Description	Usage
SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - calculation of margin and profit for CFD by indexes	SymbolInfoInteger
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - calculation of margin and profit for CFD at leverage trading	SymbolInfoInteger
SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - calculation of margin and profit for trading	SymbolInfoInteger

Constant	Description	Usage
	futures contracts on a stock exchange	
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode – calculation of margin and profit for trading futures contracts on FORTS.	SymbolInfoInteger
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode – calculation of margin and profit for trading securities on a stock	SymbolInfoInteger

Constant	Description	Usage
	exchange	
SYMBOL_CALC_MODE_FOREX	Forex mode - calculation of profit and margin for Forex	SymbolInfoInteger
SYMBOL_CALC_MODE_FUTURES	Futures mode - calculation of margin and profit for futures	SymbolInfoInteger
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - a symbol is used as a non-tradable asset on a trading account.	SymbolInfoInteger
SYMBOL_CURRENCY_BASE	Basic currency of	SymbolInfoString

Constant	Description	Usage
	a symbol	
SYMBOL_CURRENCY_MARGIN	Margin currency	SymbolInfoString
SYMBOL_CURRENCY_PROFIT	Profit currency	SymbolInfoString
SYMBOL_DESCRIPTION	Symbol description	SymbolInfoString
SYMBOL_DIGITS	Digits after a decimal point	SymbolInfoInteger
SYMBOL_EXPIRATION_DAY	The order is valid till the end of the day	SymbolInfoInteger
SYMBOL_EXPIRATION_GTC	The order is valid during the unlimited time period, until it is explicitly	SymbolInfoInteger

Constant	Description	Usage
	canceled	
SYMBOL_EXPIRATION_MODE	Flags of allowed order expiration modes	SymbolInfoInteger
SYMBOL_EXPIRATION_SPECIFIED	The expiration time is specified in the order	SymbolInfoInteger
SYMBOL_EXPIRATION_SPECIFIED_DAY	The expiration date is specified in the order	SymbolInfoInteger
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	SymbolInfoInteger
SYMBOL_FILLING_FOK	This policy means that	SymbolInfoInteger

Constant	Description	Usage
	a deal can be executed only with the specified volume. If the necessary amount of a financial instrument is currently unavailable in the market, the order will not be executed. The required volume can be filled using several offers available on	

Constant	Description	Usage
	the market at the moment.	
SYMBOL_FILLING_IOC	In this case a trader agrees to execute a deal with the volume maximally available in the market within that indicated in the order. In case the order cannot be filled completely, the available volume of the order	SymbolInfoInteger

Constant	Description	Usage
	will be filled, and the remaining volume will be canceled. The possibility of using IOC orders is determined at the trade server.	
SYMBOL_FILLING_MODE	Flags of allowed order filling modes	SymbolInfoInteger
SYMBOL_ISIN	The name of a symbol in the ISIN system (International Securi	SymbolInfoString

Constant	Description	Usage
	ties Identification Number). The International Securities Identification Number is a 12-digit alphanumeric code that uniquely identifies a security. The presence of this symbol property is determined on the side of a trade server.	

Constant	Description	Usage
SYMBOL_LAST	Price of the last deal	SymbolInfoDouble
SYMBOL_LASTHIGH	Maximal Last of the day	SymbolInfoDouble
SYMBOL_LASTLOW	Minimal Last of the day	SymbolInfoDouble
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening a position with the volume of one lot. It is used for checking a client's assets	SymbolInfoDouble

Constant	Description	Usage
	when he or she enters the market.	
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency of the symbol, charged from one lot. It is used for checking a client's assets when his/her account	SymbolInfoDouble

Constant	Description	Usage
	state changes. If the maintenance margin is equal to 0, the initial margin is used.	
SYMBOL_OPTION_MODE	Option type	SymbolInfoInteger
SYMBOL_OPTION_MODE_EUROPEAN	European option may only be exercised on a specified date (expiration, execution date, delivery date)	SymbolInfoInteger
SYMBOL_OPTION_MODE_AMERICAN	American option may be exercised on	SymbolInfoInteger

Constant	Description	Usage
	any trading day on or before expiry. The period within which a buyer can exercise the option is specified for it	
SYMBOL_OPTION_RIGHT	Option right (Call/Put)	SymbolInfoInteger
SYMBOL_OPTION_RIGHT_CALL	A call option gives you the right to buy an asset at a specified price	SymbolInfoInteger
SYMBOL_OPTION_RIGHT_PUT	A put option gives you the right to sell	SymbolInfoInteger

Constant	Description	Usage
	an asset at a specified price	
SYMBOL_OPTION_STRIKE	The strike price of an option. The price at which an option buyer can buy (in a Call option) or sell (in a Put option) the underlying asset, and the option seller is obliged to sell or buy the appropriate amount of the	SymbolInfoDouble

Constant	Description	Usage
	underlying asset.	
SYMBOL_ORDER_LIMIT	Limit orders are allowed (Buy Limit and Sell Limit)	SymbolInfoInteger
SYMBOL_ORDER_MARKET	Market orders are allowed (Buy and Sell)	SymbolInfoInteger
SYMBOL_ORDER_MODE	Flags of allowed order types	SymbolInfoInteger
SYMBOL_ORDER_SL	Stop Loss is allowed	SymbolInfoInteger
SYMBOL_ORDER_STOP	Stop orders are allowed (Buy Stop and Sell Stop)	SymbolInfoInteger
SYMBOL_ORDER_STOP_LIMIT	Stop-limit orders are	SymbolInfoInteger

Constant	Description	Usage
	allowed (Buy Stop Limit and Sell Stop Limit)	
SYMBOL_ORDER_TP	Take Profit is allowed	SymbolInfoInteger
SYMBOL_PATH	Path in the symbol tree	SymbolInfoString
SYMBOL_POINT	Symbol point value	SymbolInfoDouble
SYMBOL_SELECT	Symbol is selected in Market Watch	SymbolInfoInteger
SYMBOL_SESSION_AW	Average weighted price of the current session	SymbolInfoDouble
SYMBOL_SESSION_BUY_ORDERS	Number of Buy orders at the	SymbolInfoInteger

Constant	Description	Usage
	moment	
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Current volume of Buy orders	SymbolInfoDouble
SYMBOL_SESSION_CLOSE	Close price of the current session	SymbolInfoDouble
SYMBOL_SESSION_DEALS	Number of deals in the current session	SymbolInfoInteger
SYMBOL_SESSION_INTEREST	Summary open interest	SymbolInfoDouble
SYMBOL_SESSION_OPEN	Open price of the current session	SymbolInfoDouble
SYMBOL_SESSION_PRICE_LIMIT_MAX	Maximal price of the current session	SymbolInfoDouble

Constant	Description	Usage
SYMBOL_SESSION_PRICE_LIMIT_MIN	Minimal price of the current session	SymbolInfoDouble
SYMBOL_SESSION_PRICE_SETTLEMENT	Settlement price of the current session	SymbolInfoDouble
SYMBOL_SESSION_SELL_ORDERS	Number of Sell orders at the moment	SymbolInfoInteger
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Current volume of Sell orders	SymbolInfoDouble
SYMBOL_SESSION_TURNOVER	Summary turnover of the current session	SymbolInfoDouble
SYMBOL_SESSION_VOLUME	Summary volume of current sessio	SymbolInfoDouble

Constant	Description	Usage
	n deals	
SYMBOL_SPREAD	Spread value in points	SymbolInfoInteger
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	SymbolInfoInteger
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	SymbolInfoInteger
SYMBOL_SWAP_LONG	Long swap value	SymbolInfoDouble
SYMBOL_SWAP_MODE	Swap calculation model	SymbolInfoInteger
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Swaps are charged in money, in client deposit	SymbolInfoInteger

Constant	Description	Usage
	currency	
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Swaps are charged in money in margin currency of the symbol	SymbolInfoInteger
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Swaps are charged in money in base currency of the symbol	SymbolInfoInteger
SYMBOL_SWAP_MODE_DISABLED	Swaps disabled (no swaps)	SymbolInfoInteger
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Swaps are charged as the specified annual interest from the	SymbolInfoInteger

Constant	Description	Usage
	instrument price at calculation of swap (standard bank year is 360 days)	
SYMBOL_SWAP_MODE_INTEREST_OPEN	Swaps are charged as the specified annual interest from the open price of position (standard bank year is 360 days)	SymbolInfoInteger
SYMBOL_SWAP_MODE_POINTS	Swaps are charged in points	SymbolInfoInteger
SYMBOL_SWAP_MODE_REOPEN_BID	Swaps are charg	SymbolInfoInteger

Constant	Description	Usage
	ed by reopening positions. At the end of a trading day the position is closed. Next day it is reopened by the current Bid price +/- specified number of points (parameters SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)	
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Swaps are charged by reope	SymbolInfoInteger

Constant	Description	Usage
	ning positions. At the end of a trading day the position is closed. Next day it is reopened by the close price +/- specified number of points (parameters SYMBOL_SWAP_LONG and SYMBOL_SWAP_SHORT)	
SYMBOL_SWAP_ROLLOVER3DAYS	Day of week to charge 3 days swap	SymbolInfoInteger

Constant	Description	Usage
	rollover	
SYMBOL_SWAP_SHORT	Short swap value	SymbolInfoDouble
SYMBOL_TICKS_BOOKDEPTH	Maximal number of requests shown in Depth of Market. For symbols that have no queue of requests, the value is equal to zero.	SymbolInfoInteger
SYMBOL_TIME	Time of the last quote	SymbolInfoInteger
SYMBOL_TRADE_CALC_MODE	Contract price calculation mode	SymbolInfoInteger
SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	SymbolInfoDouble

Constant	Description	Usage
SYMBOL_TRADE_EXECUTION_EXCHANGE	Exchange execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_MARKET	Market execution	SymbolInfoInteger
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request	SymbolInfoInteger
SYMBOL_TRADE_EXEMODE	Deal execution mode	SymbolInfoInteger
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	SymbolInfoInteger
SYMBOL_TRADE_MODE	Order execution type	SymbolInfoInteger
SYMBOL_TRADE_MODE_CLOSEONLY	Allowed only position close operations	SymbolInfoInteger
SYMBOL_TRADE_MODE_DISABLED	Trade is disabl	SymbolInfoInteger

Constant	Description	Usage
	ed for the symbol	
SYMBOL_TRADE_MODE_FULL	No trade restrictions	SymbolInfoInteger
SYMBOL_TRADE_MODE_LONGONLY	Allowed only long positions	SymbolInfoInteger
SYMBOL_TRADE_MODE_SHORTONLY	Allowed only short positions	SymbolInfoInteger
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to place Stop orders	SymbolInfoInteger
SYMBOL_TRADE_TICK_SIZE	Minimal price change	SymbolInfoDouble
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK	SymbolInfoDouble

Constant	Description	Usage
	_VALUE_PROFIT	
SYMBOL_TRADE_TICK_VALUE_LOSS	Calculated tick price for a losing position	SymbolInfoDouble
SYMBOL_TRADE_TICK_VALUE_PROFIT	Calculated tick price for a profitable position	SymbolInfoDouble
SYMBOL_VOLUME	Volume of the last deal	SymbolInfoInteger
SYMBOL_VOLUME_LIMIT	Maximum allowed aggregate volume of an open position and pending orders in one direction (buy	SymbolInfoDouble

Constant	Description	Usage
	or sell) for the symbol. For example, with the limitation of 5 lots, you can have an open buy position with the volume of 5 lots and place a pending order Sell Limit with the volume of 5 lots. But in this case you cannot place a Buy	

Constant	Description	Usage
	Limit pending order (since the total volume in one direction will exceed the limitation) or place Sell Limit with the volume more than 5 lots.	
SYMBOL_VOLUME_MAX	Maximal volume for a deal	SymbolInfoDouble
SYMBOL_VOLUME_MIN	Minimal volume for a deal	SymbolInfoDouble
SYMBOL_VOLUME_STEP	Minimal volume change step for deal	SymbolInfoDouble

Constant	Description	Usage
	execution	
SYMBOL_VOLUMEHIGH	Maximal day volume	SymbolInfoInteger
SYMBOL_VOLUMELOW	Minimal day volume	SymbolInfoInteger
TENKANSEN_LINE	Tenkan-sen line	Indicators Lines
TERMINAL_BUILD	The client terminal build number	TerminalInfoInteger
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	TerminalInfoInteger
TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on	TerminalInfoString

Constant	Description	Usage
	a computer	
TERMINAL_COMMUNITY_ACCOUNT	The flag indicates the presence of MQL5.community authorization data in the terminal	TerminalInfoInteger
TERMINAL_COMMUNITY_BALANCE	Balance in MQL5.community	TerminalInfoDouble
TERMINAL_COMMUNITY_CONNECTION	Connection to MQL5.community	TerminalInfoInteger
TERMINAL_COMPANY	Company name	TerminalInfoString
TERMINAL_CONNECTED	Connection to a trade server	TerminalInfoInteger
TERMINAL_CPU_CORES	The numb	TerminalInfoInteger

Constant	Description	Usage
	er of CPU cores in the system	
TERMINAL_DATA_PATH	Folder in which terminal data are stored	TerminalInfoString
TERMINAL_DISK_SPACE	Free disk space for the MQL5\Files folder of the terminal (agent), MB	TerminalInfoInteger
TERMINAL_DLLS_ALLOWED	Permission to use DLL	TerminalInfoInteger
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the	TerminalInfoInteger

Constant	Description	Usage
	terminal settings	
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	TerminalInfoInteger
TERMINAL_LANGUAGE	Language of the terminal	TerminalInfoString
TERMINAL_MAXBARS	The maximal bars count on the chart	TerminalInfoInteger
TERMINAL_MEMORY_AVAILABLE	Free memory of the terminal (agent) process, MB	TerminalInfoInteger

Constant	Description	Usage
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, MB	TerminalInfoInteger
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal (agent), MB	TerminalInfoInteger
TERMINAL_MEMORY_USED	Memory used by the terminal (agent), MB	TerminalInfoInteger
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID data for Push notifications	TerminalInfoInteger
TERMINAL_NAME	Terminal	TerminalInfoString

Constant	Description	Usage
	name	
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to smartphone	TerminalInfoInteger
TERMINAL_OPENCL_SUPPORT	The version of the supported OpenCL in the format of 0x00010002 = 1.2. "0" means that OpenCL is not supported	TerminalInfoInteger
TERMINAL_PATH	Folder from which the terminal is started	TerminalInfoString
TERMINAL_PING_LAST	The last known value of a	TerminalInfoInteger

Constant	Description	Usage
	ping to a trade server in microseconds. One second comprises of one million microseconds	
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI).	TerminalInfoInteger
TERMINAL_TRADE_ALLOWED	Permission to trade	TerminalInfoInteger
TERMINAL_X64	Indication of the "64-	TerminalInfoInteger

Constant	Description	Usage
	bit terminal"	
THURSDAY	Thursday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
TRADE_ACTION_DEAL	Place a trade order for an immediate execution with the specified parameters (market order)	MqlTradeRequest
TRADE_ACTION_MODIFY	Modify the parameters of the order placed previously	MqlTradeRequest
TRADE_ACTION_PENDING	Place a trade order for the execution under specified condit	MqlTradeRequest

Constant	Description	Usage
	ions (pending order)	
TRADE_ACTION_REMOVE	Delete the pending order placed previously	MqlTradeRequest
TRADE_ACTION_SLTP	Modify Stop Loss and Take Profit values of an opened position	MqlTradeRequest
TRADE_RETCODE_CANCEL	Request canceled by trader	MqlTradeResult
TRADE_RETCODE_CLIENT_DISABLES_AT	Autotrading disabled by client terminal	MqlTradeResult
TRADE_RETCODE_CONNECTION	No connection with the trade server	MqlTradeResult

Constant	Description	Usage
TRADE_RETCODE_DONE	Request completed	MqlTradeResult
TRADE_RETCODE_DONE_PARTIAL	Only part of the request was completed	MqlTradeResult
TRADE_RETCODE_ERROR	Request processing error	MqlTradeResult
TRADE_RETCODE_FROZEN	Order or position frozen	MqlTradeResult
TRADE_RETCODE_INVALID	Invalid request	MqlTradeResult
TRADE_RETCODE_INVALID_EXPIRATION	Invalid order expiration date in the request	MqlTradeResult
TRADE_RETCODE_INVALID_FILL	Invalid order filling type	MqlTradeResult
TRADE_RETCODE_INVALID_ORDER	Incorrect or prohibited	MqlTradeResult

Constant	Description	Usage
	order type	
TRADE_RETCODE_INVALID_PRICE	Invalid price in the request	MqlTradeResult
TRADE_RETCODE_INVALID_STOPS	Invalid stops in the request	MqlTradeResult
TRADE_RETCODE_INVALID_VOLUME	Invalid volume in the request	MqlTradeResult
TRADE_RETCODE_LIMIT_ORDERS	The number of pending orders has reached the limit	MqlTradeResult
TRADE_RETCODE_LIMIT_VOLUME	The volume of orders and positions for the symbol has reached the limit	MqlTradeResult

Constant	Description	Usage
TRADE_RETCODE_LOCKED	Request locked for processing	MqlTradeResult
TRADE_RETCODE_MARKET_CLOSED	Market is closed	MqlTradeResult
TRADE_RETCODE_NO_CHANGES	No changes in request	MqlTradeResult
TRADE_RETCODE_NO_MONEY	There is not enough money to complete the request	MqlTradeResult
TRADE_RETCODE_ONLY_REAL	Operation is allowed only for live accounts	MqlTradeResult
TRADE_RETCODE_ORDER_CHANGED	Order state changed	MqlTradeResult
TRADE_RETCODE_PLACED	Order placed	MqlTradeResult
TRADE_RETCODE_POSITION_CLOSED	Position with the	MqlTradeResult

Constant	Description	Usage
	specified POSITION_IDENTIFIER has already been closed	
TRADE_RETCODE_PRICE_CHANGED	Prices changed	MqlTradeResult
TRADE_RETCODE_PRICE_OFF	There are no quotes to process the request	MqlTradeResult
TRADE_RETCODE_REJECT	Request rejected	MqlTradeResult
TRADE_RETCODE_REQUOTE	Requote	MqlTradeResult
TRADE_RETCODE_SERVER_DISABLES_AT	Autotrading disabled by server	MqlTradeResult
TRADE_RETCODE_TIMEOUT	Request canceled by timeout	MqlTradeResult
TRADE_RETCODE_TOO_MANY_REQUESTS	Too frequent	MqlTradeResult

Constant	Description	Usage
	requests	
TRADE_RETCODE_TRADE_DISABLED	Trade is disabled	MqlTradeResult
TRADE_TRANSACTION_DEAL_ADD	Adding a deal to the history. The action is performed as a result of an order execution or performing operations with an account balance.	MqlTradeTransaction
TRADE_TRANSACTION_DEAL_DELETE	Deleting a deal from the history. There may be cases when a	MqlTradeTransaction

Constant	Description	Usage
	previously executed deal is deleted from a server. For example, a deal has been deleted in an external trading system (exchange) where it was previously transferred by a broker.	
TRADE_TRANSACTION_DEAL_UPDATE	Updating a deal in the history. There may be cases	[MqlTradeTransaction]()

Constant	Description	Usage
	when a previously executed deal is changed on a server. For example, a deal has been changed in an external trading system (exchange) where it was previously transferred by a broker.	
TRADE_TRANSACTION_HISTORY_ADD	Adding an order to the history as a result of execu	MqlTradeTransaction

Constant	Description	Usage
	tion or cancellation.	
TRADE_TRANSACTION_HISTORY_DELETE	Deleting an order from the orders history. This type is provided for enhancing functionality on a trade server side.	MqlTradeTransaction
TRADE_TRANSACTION_HISTORY_UPDATE	Changing an order located in the orders history. This type is provided for enhancing functionality on a trade	MqlTradeTransaction

Constant	Description	Usage
	server side.	
TRADE_TRANSACTION_ORDER_ADD	Adding a new open order.	MqlTradeTransaction
TRADE_TRANSACTION_ORDER_DELETE	Removing an order from the list of the open ones. An order can be deleted from the open ones as a result of setting an appropriate request or execution (filling) and moving to the history.	MqlTradeTransaction

Constant	Description	Usage
TRADE_TRANSACTION_ORDER_UPDATE	Updating an open order. The updates include not only evident changes from the client terminal or a trade server sides but also changes of an order state when setting it (for example, transition from ORDER_STATE_STARTED to ORDER_STATE_PLACE	MqlTradeTransaction

Constant	Description	Usage
	D or from ORDER_STATE_PLACED to ORDER_STATE_PARTIAL, etc.).	
TRADE_TRANSACTION_POSITION	Changing a position not related to a deal execution. This type of transaction shows that a position has been changed on a trade server side. Position volume, open price, Stop Loss	MqlTradeTransaction

Constant	Description	Usage
	and Take Profit levels can be changed. Data on changes are submitted in MqlTradeTransaction structure via OnTradeTransaction handler. Position change (adding, changing or closing), as a result of a deal execution, does not lead to the	

Constant	Description	Usage
	occurrence of TRADE_TRANSACTION_POSITION transaction.	
TRADE_TRANSACTION_REQUEST	Notification of the fact that a trade request has been processed by a server and processing result has been received. Only type field (trade transaction type) must be analyzed for such transactions in	MqlTradeTransaction

Constant	Description	Usage
	MqlTradeTransaction structure. The second and third parameters of OnTradeTransaction (request and result) must be analyzed for additional data.	
TUESDAY	Tuesday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
TYPE_BOOL	bool	MqlParam
TYPE_CHAR	char	MqlParam
TYPE_COLOR	color	MqlParam
TYPE_DATETIME	datetime	MqlParam
TYPE_DOUBLE	double	MqlParam
TYPE_FLOAT	float	MqlParam
TYPE_INT	int	MqlParam
TYPE_LONG	long	MqlParam

Constant	Description	Usage
TYPE_SHORT	short	MqlParam
TYPE_STRING	string	MqlParam
TYPE_UCHAR	uchar	MqlParam
TYPE_UINT	uint	MqlParam
TYPE_ULONG	ulong	MqlParam
TYPE_USHORT	ushort	MqlParam
UCHAR_MAX	Maximal value, which can be represented by uchar type	Numerical Type Constants
UINT_MAX	Maximal value, which can be represented by uint type	Numerical Type Constants
ULONG_MAX	Maximal value, which can be represented by ulong type	Numerical Type Constants
UPPER_BAND	Upper limit	Indicators Lines

Constant	Description	Usage
UPPER_HISTOGRAM	Upper histogram	Indicators Lines
UPPER_LINE	Upper line	Indicators Lines
USHORT_MAX	Maximal value, which can be represented by ushort type	Numerical Type Constants
VOLUME_REAL	Trade volume	Price Constants
VOLUME_TICK	Tick volume	Price Constants
WEDNESDAY	Wednesday	SymbolInfoInteger, SymbolInfoSessionQuote, SymbolInfoSessionTrade
WHOLE_ARRAY	Means the number of items remaining until the end of the array, i.e., the entire array will be processed	Other Constants

Constant	Description	Usage
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	Other Constants